Incremental Semantics and Interactive Syntactic Processing

Nicholas John Haddock

PhD

University of Edinburgh

1988

Declaration

I declare that this thesis has been composed by myself and that the research reported therein has been conducted by myself unless otherwise stated.

Nicholas Haddock

10th October 1988.

Acknowledgements

First of all, I would like to thank my supervisors, Mark Steedman and Henry Thompson. Mark Steedman has provided constant encouragement and support in all matters, and I am deeply indebted to him for the thoughtful guidance and attention he has given me over the years. I am grateful to Henry Thompson for many stimulating supervisions, and the broad perspective he brought to every aspect of the project.

I owe a great deal to my friends and colleagues in Edinburgh. Jo Calder, Robert Dale, Alex Lascarides, Colin Matheson, and Jon Oberlander have all helped in ways too numerous to mention, and working with them has been a lot of fun. I would also like to thank Einar Jowsey, Ewan Klein, Chris Mellish, Marc Moens, Remo Pareschi, Bonnie Webber, and Mary Wood for their advice. I am especially grateful to Gerry Altmann for lively discussion of all things referential, and to Glyn Morrill and Mark Hepple for many sound and complete conversations about parsing.

I would like to say a special thank you to my mother and father, who have been a very deep source of support during these studies.

The financial help of an SERC Research Studentship is gratefully acknowledged. I would also like to thank the University of Edinburgh and Hewlett-Packard Laboratories for providing the time and financial support necessary to complete the work.

Abstract

This thesis starts from an observation that is often made about human language understanding: that the comprehension of a sentence appears to proceed in a piecemeal, "incremental" fashion, carried out in small, gradual steps as each word is encountered. The thesis is computational in orientation, and aims to develop a model of linguistic processing which is capable of such incremental interpretation.

It is argued that a number of previous attempts to formulate a process of incremental semantics fall short of providing a truly left-to-right, word-by-word, incremental interpretation procedure. Given that these models view semantic analysis as a syntactically driven process, this general shortcoming can be attributed in part to their syntactic components. To a greater or lesser extent, these models do not perform incremental syntactic analysis.

With this in mind we import Steedman's theory of extended categorial grammar as a basis for syntactic processing, since it is directly compatible with the hypothesis of incremental interpretation.

The substance of the thesis concerns noun phrases, and how they can be incrementally interpreted with respect to the prevailing referential context. Here we employ Mackworth's network consistency techniques (otherwise known as "constraint satisfaction" or "Waltz filtering"), since, as Mellish has shown, they offer an inherently incremental process of reference evaluation. The thesis reviews these techniques, and goes on to investigate their formal adequacy for the task of reference evaluation, exploiting Freuder's graph-theoretical work on constraint satisfaction.

Having done this, we integrate network consistency operations with the syntactic processing to produce a model of incremental reference evaluation. The model is then applied to two specific problems in natural language: a particular problem in the semantics of definite descriptions, and the ubiquitous problem of syntactic ambiguity. In the latter case, we see how the process of incremental reference to the context acts to resolve certain syntactic ambiguities as they are discovered by the overseeing syntactic parser. The model thus predicts that local syntactic ambiguities are often resolved by contextual factors, in accordance with the psycholinguistic theory of sentence processing advocated by Crain and Steedman.

Contents

Chapter 1

Introduction

1.1. Incremental Interpretation and its Implications

Our intuitions about natural language comprehension strongly suggest that the process of understanding a spoken or written sentence is a continuous one, carried out in small, gradual steps as each word is heard or read. We might therefore think of the comprehension process as being "incremental", in that an utterance seems to acquire its meaning bit by bit rather than in one go when it has come to an end. Furthermore, we have the feeling that something deeper than just the sense of a word or phrase is accessed "on-line". It seems that knowledge specific to the context, such as referential information, also becomes available immediately — at least if we are to trust our intuitions.

But we have more to go on than simple introspection. A number of psycholinguistic inquiries have provided evidence which lends support to this hypothesis. These have concerned the incremental, real-time nature of comprehension (Marslen-Wilson, 1975; Tyler and Marslen-Wilson, 1977; Marslen-Wilson and Tyler, 1980; Marslen-Wilson, 1987; Tanenhaus, Carlson and Seidenberg, 1985), the immediacy of reference resolution (Shillcock, 1982; Garrod and Sanford, 1985), and the early context-dependent disambiguation of ambiguous nouns (Swinney, 1979). It thus seems reasonable to suppose that incremental interpretation is indeed a feature of the human language processing faculty.

The present work is motivated principally by the desire for a computational model of sentence processing which reflects this basic psychological observation. Much computational work in natural language understanding flies in the face of the evidence, postulating distinct processing stages in which syntactic, semantic and pragmatic analysis are performed in sequence, each component completing its work before handing control over to the next one. To take an early and influential example, the LUNAR system of Woods, Kaplan and Nash-Webber (1972) postpones semantic processing until the first complete syntactic parse of the input sentence becomes available. A semantic component then transforms the syntactic structure into a corresponding semantic translation. Of course, this kind of sequential processing is usually motivated by engineering considerations, such as its relative efficiency over designs where semantics and world knowledge are considered during the

parsing process. (Though naturally the relative economies of alternative architectures will depend on the application in question.) But from what we know of the human sentence processor, the sequential view of comprehension is psychologically unrealistic. A more probable design would interleave syntactic and semantic analysis, incrementally building up a semantic representation in a single left-to-right sweep of the input.

The aim of this thesis is to develop and investigate one such computational model of incremental semantics. The general motivation is that a sentence processing program embodying incremental interpretation might be interesting from a psychological point of view, and so contribute to research in cognitive science. However, we can expect more than a certain amount of psychological plausibility from this approach; the hypothesis that sentences are incrementally interpreted also appears to have implications for theories of natural language and natural language processing.

For example, if semantic interpretations are available at every turn in sentence processing, then there is every reason to suppose that the local syntactic ambiguities which abound in natural language sentences are resolved by taking into account their appropriateness in the context of utterance. The psycholinguistic experiments of Crain and Steedman (1985) and Altmann and Steedman (1988) provide strong support for this position. Indeed, if the human parser did not make frequent appeal to an incrementally evaluating semantic component, it would be hard to explain how natural language could tolerate such vast amounts of ambiguity and why we are so rarely aware of it in ordinary discourse.

Incremental interpretation may also be a fundamental factor in the nature of natural language semantics itself. The meaning of many natural language categories seems to have an inherent dynamic or procedural quality, whereby the force of an expression is intimately bound up with the prevailing linguistic and non-linguistic context. Pronominal reference provides an obvious example of this phenomenon, as evidenced by the following instance of intra-sentential anaphora:

(1) *Some students* have left *their* bags

Why is it possible for the italicised expressions in (1) to co-refer? A promising explanation is that the initial indefinite noun phrase *some students* changes the context, introducing or emphasising a group of students, and so provides a reference for the subsequent pronoun. As we would expect with an incrementally interpreting processor, the order of the referring expressions is significant; notice that we do not say

(2) * *Their* bags have been left by *some students*

without having first changed the context appropriately. Certainly there are cases like (3)

(3) Near *him*, *Dan* saw a snake

where an anaphor (*him*) refers to an entity (*Dan*) mentioned later in the string. But such instances of so-called "backwards anaphora" are comparatively rare, and tend to be restricted to specific kinds of syntactic construction; the overwhelming tendency of pronouns is to refer to entities already mentioned in the discourse. As Isard (1975) illustrates, many other varieties of reference (for example, demonstrative reference and temporal reference) can be seen as a reflection of the procedural, left-to-right constraints imposed by an incrementally interpreting sentence processor.

Thus a computational model embodying incremental semantics would not only be psychologically attractive, but might also provide a suitable platform on which to investigate some fundamental features of natural language. With this in mind, we now turn to the specific goals of the present research.

1.2. The Aim of the Thesis

The specific aim of this thesis is to develop a model of incremental reference evaluation, concentrating on full singular noun phrases like *the green apple in the bowl*. Very little is known about the psychological process of reference resolution, though our own experience as language users and informal experiments by Cohen (reported in Goodman, 1986) suggest that some form of referential processing can occur before the end of a noun phrase.[1] Following Mellish (1985), we will model incremental noun phrase evaluation using a type of constraint satisfaction process known as network consistency. A tangential goal is to review the network consistency paradigm and investigate its formal adequacy for the task of reference evaluation.

To make our goals a little more concrete, the model will be applied to two particular linguistic problems where we should expect incremental interpretation to be of help. The first of these concerns a long-standing problem in the semantics of definite descriptions. Suppose we have a context (pictured in Figure 1.1) in which there are three rabbits, two hats and one box, where one rabbit is in a hat and one is in the box. Assuming that both

[1] Cohen gave two people (one skilled, the other non-skilled) the task of assembling a toy water pump from its various components — tubes, nozzles, valves, and so on, all of different colours, shapes and sizes. The domain provided a rich source of perceptual information, evoking complex referring expressions from the expert and apprentice in their task-oriented dialogue. Studying videotape excerpts, Cohen noticed that on certain occasions when the expert asked for a particular object (perhaps with an expression like *the red thing with prongs on it*), the apprentice would

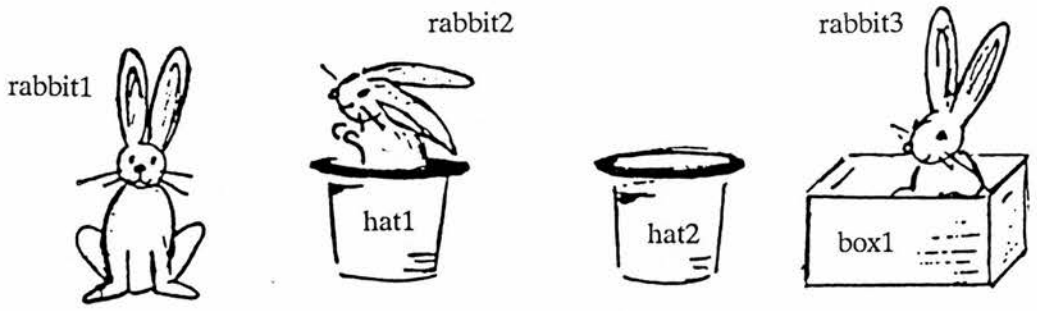


Figure 1.1.

speaker and hearer are aware of this situation, it would be acceptable to refer to *rabbit2* by means of the following complex NP:

(4) the rabbit in the hat

Consider the expression being uttered without any previous talk of rabbits or hats. What is interesting about it is the use of the definite determiner in the inner, simple NP, *the hat*. As a whole, the expression in (4) sounds perfectly natural, and so suggests that a unique hat is identifiable to the hearer. But given that there are two hats, and not one, in the scene, why is the definite permissible?

The answer to this question seems obvious. Although there are two hats to think of, the expression demands that the hat in question must have a rabbit in it, and there is only one such hat in the situation. However, any strictly compositional account of NP semantics, such as Winograd (1972) or Hirst (1987), would ignore this information and so judge (4) to be infelicitous. Such theories would require the somewhat convoluted version in (5), if a definite is to be used at all to specify the hat.

(5) # the rabbit in the hat with a rabbit in it

A semantics organised around the criterion of incremental interpretation should have less trouble with (4). If we assume that a hearer incrementally evaluates a semantic representation — after each word, say — the empty hat in the scene will never really be considered a viable candidate for the inner NP. When the word *rabbit* is reached, a hearer can collect together in his or her mind the set of rabbits in the context. After the preposition, this set can be refined to containing only rabbits which are *in* something and, most importantly, the

hearer can start thinking about another set of objects, those which have rabbits in them. There is only one hat in this new set and so by the time the inner NP is processed a definite determiner sounds natural.

The thesis will also illustrate the way in which a syntactic parser can incrementally *interact* with the process of reference evaluation and so resolve certain structural ambiguities. Crain and Steedman (1985) and Altmann and Steedman (1988) present convincing evidence (to which we will return in Chapter 2) that the immediate referential context is of paramount importance to the human sentence processor in resolving local and global structural ambiguities, such as the ambiguous attachment of the prepositional phrase in (6):

(6) John saw the boy with the telescope

In structural terms, the prepositional phrase *with the telescope* may either modify the noun *boy* or the verb *saw*. However, Crain and Steedman (1985) suggest that the local, referential context will often resolve such ambiguities, since the human sentence processor will take the reading which makes most sense in the context of utterance. So if there are two boys under discussion, and one of these is associated with a telescope, the noun-modifying reading of (6) is assumed because this yields a unique boy, as the speaker presumably intended by using a definite article.

As Altmann and Steedman (1988) note, such attachment decisions can be made early in the parse of a sentence. In (6), for example, if after reading *boy* the noun phrase *the boy* fails to refer to a unique contextual entity, the processor can at once expect that the subsequent material will modify this expression. The computational model of noun phrase interpretation will therefore be applied to ambiguous noun phrases, and its specific predictions for ambiguity resolution analysed in the light of data concerning human preferences for one reading over another.

So while the primary purpose of the thesis is to provide a well-specified model of incremental reference evaluation because of its own intrinsic interest, the thesis has two secondary aims: to present a novel solution to a problem of definite reference, and to present an account of how certain local and global ambiguities may be incrementally resolved by continual reference to the context during parsing. As Chapter 2 shows, existing models of sentence processing are deficient in this respect.

---

sometimes initiate a physical search for the object before the referring expression was complete.

1.3. The Scope of the Work

Before moving on to the substance of the thesis, it will be helpful to consider the scope of the approach to reference, and in particular the ways in which it is limited. Firstly, we will consider only full, singular noun phrases such as those alluded to above. Hence more abbreviated referring expressions, like pronouns and ellipses, are not addressed; and all forms of plural reference are similarly ignored.

We will simplify even further in our characterisation of the domain of reference. Here we will just make do with a database, or set of facts, indicating which entities are available for reference. For example, the situation pictured in Figure 1.1 is represented as follows:

(7) { rabbit(rabbit1), rabbit(rabbit2), rabbit(rabbit3),
hat(hat1), hat(hat2), box(box1),
in(rabbit2,hat1), in(rabbit3,box1)}

(7) indicates that {rabbit1, rabbit2, rabbit3, hat1, hat2, box1} are the entities available for reference, that *hat1* contains *rabbit2*, and so on. Sets such as (7), which we will call *contexts*, are designed to represent a hearer's view or "mental model" of the domain of reference. This is a somewhat idealised way of characterising the human view of the available entities. At any given point, a speaker of natural language may refer to any one of a vast number of complex entities and concepts, and so in reality the domain of reference is the entire world, rather than small, finite sets such as (7). However, the hope is that our small, finite contexts and the procedures that will access them can at least be seen as a subset of the structures and processes in a general model of reference.

A related point is that we will not be concerned with how the various contextual entities and properties become prominent in the hearer's mind in the first place. The present work is thus largely orthogonal to that of Grosz (1977), Webber (1978), and Sidner (1979), which has studied the way in which entities are introduced to the discourse and how the focus of attention among these entities shifts as the discourse proceeds. It may be that contexts such as (7) are the product of the prior discourse, or simply represent an aspect of the immediate physical situation. Given the nature of the examples we consider, the latter, "perceived physical situation" use may be the more appropriate interpretation, but the issue is not a central one.

The linguistic significance of this simplification is that all expressions are taken to refer to entities already known to the hearer (and, incidentally, the speaker). Within the context of (7), then, we will assume that the definite NP *the box* refers to *box1*, and that

the indefinite noun phrase *a hat* refers, non-specifically, to either *hat1* or *hat2*. Of course, this is not the general case for either definite or indefinite noun phrases. In particular, indefinite expressions are commonly used by a speaker to *introduce* entities into the discourse. In this respect, then, the treatment of indefinite reference will be extremely restricted. However, we should note that there are certainly situations where indefinites are used in this limited way; for example, in imperative constructions such as

(8) Pick up a rabbit

Assuming that both speaker and hearer are aware of the facts of the situation in (7), the hearer would normally respond to the instruction in (8) by picking up one of the three rabbits. And to do this the hearer must assume that the noun phrase *a rabbit* refers (non-specifically) to an existing entity in his or her mental contextual database.[2] Note also that the thesis will have nothing to say about non-restrictive noun-modifiers, and will similarly ignore the referential/attributive distinction (cf. Donnellan, 1966).

Finally, given our limited view of semantic interpretation, the program described below is organised as a processor of noun phrases rather than sentences.

1.4. An Outline of the Thesis

The remainder of the thesis runs as follows:

The next chapter is divided into two parts. The first half reviews psycholinguistic theories of sentence processing, concentrating on Crain and Steedman's (1985) and Altmann and Steedman's (1988) proposal that the human syntactic processor interacts with semantic and pragmatic components on a possibly word-by-word basis during parsing. The present work can be viewed as an attempt to computationally formalise aspects of their theory, a theory which emphasises the systematic use of local contextual knowledge by the human sentence processing mechanism ("HSPM" for short).

The second part of Chapter 2 considers previous computational work on incremental semantics and interactive syntactic processing. We devote most attention to the respective models of Bobrow and Webber (1980b), Winograd (1972), and Mellish (1985). Although these programs achieve varying degrees of success in implementing an incremental and interactive semantics, it is argued that they all fall short of providing a truly left-to-right, word-by-word, incremental interpretation procedure. A final section examines this general shortcoming and concludes that it is symptomatic of a common problem of grammar. All

the above programs, and those in their shadows, assume more or less conventional theories of syntax which are much better suited to processing from right to left than from left to right.

With this in mind, Chapter 3 turns to a syntactic framework which is more compatible with the incremental interpretation hypothesis, Steedman's (1987b) theory of "combinatory" categorial grammar. We see how the grammatical machinery of combinatory grammar, which Steedman introduces to capture facts about discontiguous dependency, provides a ready-made syntactic foundation for incremental processing. We also review a simple bottom-up, shift-reduce parsing technique for analysing input strings with respect to the grammar.

Chapter 4 is also foundational, in that it presents the theoretical basis for noun phrase evaluation. Here we adopt Mellish's (1985) proposal to employ network consistency techniques (otherwise known as "constraint satisfaction" or "Waltz filtering"), since they provide an inherently incremental process of refinement, suitable for gradually resolving a reference as semantic constraints are imposed. The chapter reviews these methods, and goes on to investigate their formal adequacy for the task of reference evaluation.

Chapter 5 is the central chapter of the thesis. It ties together the two starting points of the preceding chapters, by first determining a convenient scheme of semantic translation to parallel the syntactic representation, and then relating the semantic translations to the network consistency procedure defined earlier. The result is an incremental model of processing in which every constituent, from the lexical level upwards, has three levels of description; corresponding to its syntax, its semantics, and its extension in the context. The chapter then considers the semantics of the definite article, formalising its condition of uniqueness as a "closure constraint" to be enforced when a referring expression is syntactically closed, rather than taking effect as soon as the article is read. The final section returns to the two application areas discussed in section 1.2, successfully applying the model to the problematic "rabbit" phrase of (4). We then analyse the kind of syntactic ambiguity resolution performed by the on-line process of incremental reference evaluation, and see how its behaviour accords with Crain and Steedman's and Altmann and Steedman's pragmatic parsing principles.

In Chapter 6 we re-examine the simple shift-reduce model of parsing presented in Chapter 3, and observe how a chart parser explores the syntactic search space more

[2] See Hawkins (1978, pp172-227) for further discussion.

intelligently.

A concluding chapter investigates the theoretical implications of the work for definite reference, and points to some areas for further research.

Chapter 2

Models of Incremental and Interactive Semantics

This chapter reviews work in two areas of sentence comprehension. Our principal interest will be in models which attempt some form of incremental semantic interpretation, during the process of syntactic analysis. We will also look at the closely related, though logically independent, issue of structural ambiguity resolution by interaction with an on-line semantic component.

Although this is essentially a survey of computational work, we will start with psychological models of syntactic ambiguity resolution. Here we pay most attention to the empirical and theoretical work of Crain and Steedman (1985) and Altmann and Steedman (1988), which is concerned with the effect of context on the resolution process. The present computational endeavour subscribes to their view of the HSPM, and so by reviewing this work we are setting the scene for our later account of local ambiguity resolution.

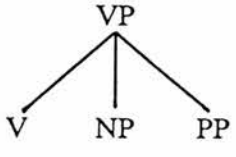
2.1. A Psychological Model of Syntactic Ambiguity Resolution

Much work in psycholinguistics has been directed towards showing that syntactic or "structural" factors are dominant in resolving local syntactic ambiguity. Kimball (1973), Frazier and Fodor (1978), Ford, Bresnan and Kaplan (1982), Rayner, Carlson and Frazier (1983), Ferreira and Clifton (1986) all argue that the HSPM employs parsing strategies based on context-independent, structural information rather than local pragmatic knowledge. One proposed processing heuristic concerns globally ambiguous sentences like

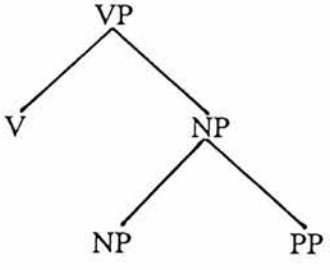
(1) She tickled the man with the gladiolus
(From Altmann and Steedman, 1988)

in which the VP, *tickled the man with the gladiolus*, has the two syntactic readings illustrated in (2) (given certain assumptions about phrase-structure):

(2) a.



b.



The analysis in (2a) characterises the PP *with the gladiolus* as an argument of the verb, while (2b) has it Chomsky-adjoined to the NP. The HSPM supposedly prefers the reading in (2a), in which the gladiolus does the tickling, and it has been proposed that this is because the syntactic analysis in (2a) requires less non-terminal nodes than (2b). This strategy of choosing the syntactically least complex reading, which Frazier (1978) calls Minimal Attachment, is purported to determine the analysis first derived by the HSPM in cases like (1) — regardless of the prevailing context. According to such theories, alternative analyses will only be attempted if the initial reading suggested by structural considerations later turns out to be incompatible with general knowledge about the world or the specific conversational context.

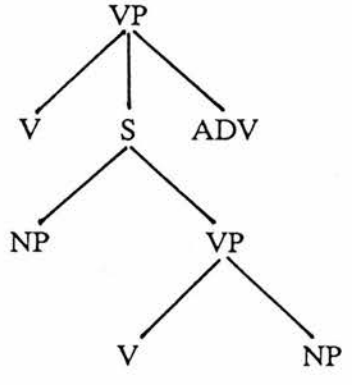
Some sentences have multiple readings which are *equivalent* in syntactic complexity. For example, consider

(3) She said it tickled her yesterday

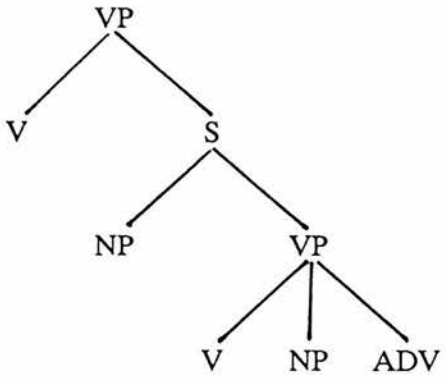
(From Altmann and Steedman, 1988)

and the two possible analyses of the VP *said it tickled her yesterday*:

(4) a.



b.



Here both phrase-structure trees involve the same number of non-terminal nodes and are therefore equivalent in complexity. However, there seems to be a preference in such examples for the second reading, portrayed in (4b), where the adverb *yesterday* modifies the tickling rather than the saying. To accommodate this data, a number of researchers have suggested that the processor attaches an incoming constituent like *yesterday* as low down and as far to the right as possible in the tree. This strategy has variously been called Right Association (Kimball 1973), Late Closure (Frazier 1978), and Final Arguments (Ford, Bresnan, and Kaplan 1982).

A related programme of research has concentrated on the form of processing architecture which might give rise to these structural parsing strategies. Thus we have the "two-

stage" Sausage Machine (Frazier and Fodor, 1978; Fodor and Frazier, 1980), arc-ordering in an ATN (Wanner, 1980), limited parallelism in an ATN (Wanner, 1987), and conflict-resolution in a shift-reduce parser (Shieber, 1983; Pereira, 1985), among others. The goal in this work has been to "explain", or to "describe" precisely, how the strategies emerge from an underlying organisation of sentence processing.

These structural approaches have also been applied to grammatically correct and unambiguous "garden path" sentences, such as

(5) The horse raced past the barn fell

from Bever (1970), so called because readers feel as if they have been led down the garden path by the reduced relative *raced past the barn*. Readers of (5) often experience confusion on reaching the word *fell* because they have taken the word *raced* as the verb of the main clause rather than as a past participial, as demanded by the global syntax of the sentence. It has been suggested that the HSPM adopts the wrong reading because it resolves the local ambiguity of *raced* by a method akin to Minimal Attachment; *raced* is read as a main verb since this requires less non-terminal nodes than the alternative.

How can we reconcile these processing strategies with our intuitions about incremental interpretation in the context, which common-sense suggests should play a significant role in local ambiguity resolution? A good place to start might be the data itself. Crain (1980) points out that it is in fact rather odd to study the effect of sentences like (5) in isolation from a relevant context, for definite noun phrases like *the horse* and *the horse raced past the barn* are generally used to pick out entities already present in the discourse. In particular, Crain noted that the alternative readings of the fragment

(6) The horse raced past the barn ...

— one involving a subject NP *the horse* and the other with subject *the horse raced past the barn* — vary not only in their syntactic complexity but also in their *presuppositional* complexity. The simple definite NP *the horse* presupposes that just one horse is established in the discourse and is available for reference, whereas the complex definite NP *the horse raced past the barn* presupposes that a *set* of horses is under discussion and, of these, one is distinguished by the property of having been raced past the barn. Crain conjectured that the HSPM would analyse a sentence such as (5) so that the presuppositions of its NPs are satisfied; with the implication that the choice of analysis would depend on the context in which the sentence is read. This, in turn, would mean that it should be possible to "control" the garden-path effect by changing the context appropriately.

To investigate this, Crain presented subjects with pairs of sentences, each with a local ambiguity after the word *that*:

(7) a. Sentence A (Sa)

The psychologist told the wife that he was having trouble with her husband.

b. Sentence B (Sb)

The psychologist told the wife that he was having trouble with to leave her husband.

The syntax of Sa requires *that* to be read as a complementiser, whereas in Sb it must be read as a relative pronoun. Each sentence was preceded by either one of the referential contexts in (8):

(8) a. Context A (Ca)

A psychologist was counselling a married couple. One member of the pair was fighting with him but the other one was nice to him.

b. Context B (Cb)

A psychologist was counselling two married couples. One of the couples was fighting with him but the other one was nice to him.

Note how the sentences in (7) relate to the contexts in (8). The correct reading of Sa demands that the direct object of *told* must be the simple NP *the wife*. This presupposes that a unique wife is available for reference, and so is supported by Ca but not Cb. In contrast, the syntax of Sb requires the direct object of the verb to be a relativised NP, and so Sb is appropriate in contexts consisting of more than one wife — that is, Cb but not Ca. Crain found that subjects did not experience difficulties with either of these sentences in (7) when they were preceded by their appropriate contexts. Moreover, when the conditions were crossed — so that Ca preceded Sb and Cb preceded Sa — *both* sentences *did* engender a garden-path effect.

This effect of context does not appear to be limited to syntactically unambiguous garden-path sentences. Altmann (1985, 1986, 1987, 1988) and Altmann and Steedman (1988; hereafter "A&S") report experiments in the same context-sentence paradigm, finding a similar effect of context on globally ambiguous materials (i.e. like (1) but unlike (5)). In addition, it appears that the specific preceding referential context can not only over-ride the purported tendency towards Minimal Attachment, but also any possible tendency to choose the reading most consistent with considerations of general "plausibility". For example, A&S preceded each of the sentences in (9) with a context including two safes; one

with a new lock and one with an old lock.

(9) a. The burglar blew open the safe with the dynamite and made off with the loot.
b. The burglar blew open the safe with the new lock and made off with the loot.

In (9a), the VP-modifying reading of *with the dynamite* is suggested both by Minimal Attachment and considerations of general plausibility, given our knowledge about dynamite and what it is used for. However, this sentence took far longer to process than (9b), the presuppositions of which are satisfied by the two-safe context. On the basis of this and other evidence, A&S conclude that the HSPM resolves attachment ambiguities by reference to the context rather than by Minimal Attachment.

The above evidence suggests that contextual factors have been underestimated in recent theories of the HSPM. If this is so, which contextually-oriented parsing strategies might govern its behaviour? Crain and Steedman (1985; hereafter "C&S") suggest a set of principles, one of which is the Principle of Referential Success:

The Principle of Referential Success

> If there is a reading which succeeds in referring to an entity already established in the hearer's mental model of the domain of discourse, then it is preferred over one that does not.
>
> (From Crain and Steedman, 1985)

This principle, inspired by Winograd (1972), accords with both Crain's and Altmann's empirical evidence suggesting that the HSPM chooses the syntactic analysis in which noun phrases refer successfully. Taken literally, though, it means that the sentence processor must wait until all possible NP readings (relating to a particular head noun) are syntactically complete before choosing between them. Altmann (1986) has observed that the decision could be made earlier if the principle were reformulated in terms of referential *failure* rather than *success*, and presents this alternative:

The Principle of Referential Failure

> If a definite referring expression fails to restrict the set of candidate referents in the hearer's mental model of the domain of discourse to a single member, then an analysis which treats subsequent material as a modifier for that referring expression (i.e. as providing information which may lead to further restriction) will be favoured over one that does not.
>
> (From Altmann, 1986)

The Principle of Referential Failure is more consistent with an incrementally interpreting processor, since it may resolve an attachment ambiguity as soon as the simple NP analysis *fails* to refer successfully. Consider, for example, the following text:

(10) A psychologist was counselling two women. He was worried about one of the pair but not about the other. The psychologist told *the woman* that ...

(From Altmann and Steedman, 1988)

For this text, which introduces two women, the principle predicts that *that* is immediately disambiguated as a relative pronoun modifying *woman*, because the simple NP *the woman* fails to refer uniquely.

More recently, A&S have combined the above principles of Referential Success and Failure into a more general principle, of "referential support":

The Principle of Referential Support

An NP analysis which is referentially supported will be favoured over one that is not.

(From Altmann and Steedman, 1988)

In this respect, A&S regard a noun phrase as "referentially supported" when all its referential presuppositions are satisfied by the context. In addition, A&S assume that the processor which employs this principle performs incremental interpretation, and thus checks for referential support word-by-word as a sentence is read. Given this assumption, this new principle clearly subsumes the earlier, specific principles of Referential Success and Failure.

Notice that the parsing principles above assume that the domain of reference for a referring expression is the hearer's *mental representation* of the world, rather than the world itself. This is an important distinction because a hearer often encounters definite expressions for which no referent exists in his or her mind, such as in explicitly intensional constructions like

(11) Did you see *the man who just walked past the window*?

(From Crain and Steedman, 1985)

In these cases, the speaker is aware of a referent for the expression, but the hearer might not be; in fact, the point of the question in (11) is to resolve this very uncertainty. In situations where no referent exists in the hearer's mental "database", the hearer must assume the presuppositions of the phrase and change his or her view of the world appropriately. Thus, in response to the question in (11), a "token" man must be introduced to the hearer's database, with the attendant properties of having walked past the window and of being unique in this respect.

It will be convenient to distinguish between the two uses of a definite referring expression contrasted above as *given* and *new* uses, respectively. (The terms are borrowed from Halliday (1967), and Clark and Haviland (1977), though we will use them here only

in an informal way.) Where an expression refers to an entity already established in the hearer's database representing his or her view of the world, we will say that the expression relates *given* information. In contrast, if an expression refers to an entity not already represented in the hearer's mind then it conveys *new* information.

C&S note this distinction and go on to explain the behaviour of the HSPM when it is processing out of context — in the so-called "null" context characteristic of many psycho-linguistic experiments — where there are no entities available for reference by definite expressions such as *the horse*. They point out that in the null context definite referring expressions always present new information. And as we have seen, the locally available readings in sentences such as (5) vary in presuppositional complexity. They argue that it seems reasonable to suppose that the HSPM chooses the locally available reading which *minimises* the number of *unsatisfied* presuppositions, and so minimises the number of alterations that have to be made to its contextual database. This idea is enshrined as the Principle of Parsimony:

> *The Principle of Parsimony*
>
> A reading which carries fewer unsatisfied presuppositions will be favoured over one that carries more.
>
> (From Altmann and Steedman, 1988)

The commonly experienced garden-path effect of (5) is now explained by the fact that the simple NP analysis *the horse* carries fewer presuppositions than the complex NP analysis *the horse raced past the barn*: the former requires the addition of only one horse to the hearer's model, whereas the latter analysis involving a restrictive modifier requires that a set of horses be introduced.

The above evidence suggests that semantic factors play a very influential role in resolving syntactic ambiguities during sentence processing. Furthermore, the data indicates that specific referential knowledge, local to the discourse, presides over general, context-independent knowledge about the world when considering the likelihood of a particular syntactic analysis. And if specific, known referents are not available at the time of processing a definite NP, C&S predict that the HSPM prefers the syntactic reading which minimises the number of referential presuppositions unsatisfied by the context.

The approach to the HSPM taken by C&S and A&S provides the overarching research programme in which the present thesis is situated. A central goal of this thesis is to produce a well-specified computational model of semantic interpretation in the context, a model in which syntactic and referential processing interact in accordance with the

psychological data reviewed above. In particular, we will concentrate on producing behaviour consistent with the Principle of Referential Support, which concerns expressions which refer to given, known entities.

Our computational model will therefore attempt to account for (a subset of) the above referential parsing principles in much the same way as the computational models of Shieber (1983), Pereira (1985) and Wanner (1987) attempt to explain the structural parsing principles of Minimal Attachment and Right Association. Naturally enough, the architecture of our resultant referential model will stand in marked contrast to that of the structural models. So whereas Marcus (1980), Milne (1982), Shieber (1983), Berwick and Weinberg (1984) and others have advocated deterministic or near-deterministic syntactic processors, like Bear (1983) and Pulman (1986) our interest will be in non-deterministic parsers which explore all syntactic analyses so that contextual factors can choose between them.

Of course, there has already been much work on sentence processors which perform semantic analysis on-line to the syntactic parser, and it is these which occupy our attention for the remainder of this chapter. However, as we will see, none of these programs implements the exact kind of interaction determined above as desirable in a model of the HSPM.

2.2. Computational Models of Incremental and Interactive Semantics

We now turn to existing computational models of sentence processing, confining our attention to those programs which perform some semantic interpretation "on-line", during syntactic parsing. Existing work in this field can be divided up and categorised in a number of ways, and the following distinctions will be helpful in our review of the research.

The first distinction concerns the way the semantic representation is constructed. Here we will be looking for an *incremental* construction process, where a processor gradually accumulates semantic information as an input sentence is read from left to right. The degree to which a given process is incremental can be determined by considering how soon the semantics of an incoming word is incorporated into the semantics of the entire preceding sentence fragment.

We may also analyse a scheme of semantic interpretation in terms of its *interaction* with the syntactic processor. In particular, how frequently does the semantic component

guide the steps of the parser? The frequency of interaction varies from system to system; in some the semantics provides feedback on a word-by-word level, and in others guidance is restricted to the points at which major syntactic constituents are completed.

A third and final distinction regards the type of *semantic information* manipulated during parsing. Here the crucial distinction is between those models which involve interactions with temporary but specific knowledge about the current conversational or physical context, and those models which only involve a more general, common-sense knowledge about what is plausible in the world (such as the knowledge which tells us that it is more likely for doctors to cure patients than the reverse). Following the preceding discussion, in which we have identified reference to the context as a desirable characteristic of an incremental interpreter, we will devote most attention to the systems which explicitly include on-line referential processing. (See Ritchie (1983) for a more general discussion of the role of semantics in parsing.)

However, the majority of systems have concentrated on the deployment of *context-independent* semantic and world knowledge during sentence processing. Winograd (1972), Woods (1973), Ford, Bresnan and Kaplan (1982), Milne (1982) and Pazzani (1984) advocate the use of shallow, sense-semantic information stemming from individual lexical items (for example, to check that the agent of the verb *drive* is animate), while Bobrow and Webber (1980b), Lytinen (1984), Waltz and Pollack (1985), Wilks, Huang and Fass (1985), Lesmo and Torasso (1985), Charniak (1986), Dahlgren and McDowell (1986), Palmer *et al* (1986), Hirst (1987) and Jacobs (1987) employ a variety of more general knowledge to guide the process of syntactic analysis. Some models, such as those of Riesbeck (1975), Burton (1976), and Tomabechi (1987), do away with a distinct syntactic level altogether, making information about word order and constituency more or less implicit in the interpretation process.

Less work has been directed at the involvement of pragmatic elements in the parsing process, although Gehrke (1983) discusses how a processor may incrementally determine a speaker's intentions in the context of an ongoing dialogue. Weischedel (1979) shows how the presuppositions and entailments of a sentence may be computed as it is parsed, but his inference procedures are limited to the structural aspects of the sentence and do not relate to the local context. More relevant from the present point of view are the systems of Winograd (1972), Mellish (1985), and Hirst (1987), which all process referential informa-

tion during syntactic parsing, and these are discussed below.[1]

However, in order to broadly characterise those systems which only access general, context-independent world knowledge during parsing, our survey will begin by reviewing an early and innovative example of incremental semantic processing which proceeds without the help of local knowledge about the discourse, Bobrow and Webber's RUS/PSI-KLONE system.

2.2.1. Incremental and Interactive Sense-Semantics

This section discusses the RUS/PSI-KLONE program developed at BBN Laboratories by Bobrow and Webber (1980a, 1980b),[2] a program which maintains the formal distinction between syntax and semantics, but allows them to interact in an incremental fashion during sentence processing. Bobrow and Webber's PSI-KLONE consists of an ATN-based syntactic parser[3] which engages in a dialogue of requests and responses with a semantic interpreter as an input sentence is scanned from left to right. As an arc in the network is traversed, it may transmit a "request" to the semantic component, typically in the form of an instruction to partially interpret the constituent being parsed. The semantic component will respond either with an updated version of the sentence's interpretation, or with a failure message indicating that the proposed syntactic move is implausible on semantic grounds. If such a failure message is received, the parser abandons that particular path through the ATN and backs up in the normal way.[4]

This allows syntax and semantics to interact to resolve syntactic and word-sense ambiguities early on in sentence processing. Consider, for example, a sentence fragment of the form in (12).

(12) Smith sent NP ...

(12) contains a local structural ambiguity: the NP may be the direct object of the verb (as

---

[1] Schubert (1984, 1986) also describes a program in which syntactic, semantic, and pragmatic factors all contribute to the disambiguation of a syntactically ambiguous sentence, but his model is too briefly described to comment on here. We will come to other relevant work, such as Pulman's (1986) incremental process of syntactic and semantic analysis, and Hobbs' (1986) work on "local pragmatics", as the thesis develops.

[2] See also Bates, Bobrow and Webber (1981), Brachman (1979), and Sondheimer, Weischedel and Bobrow (1984).

[3] For a general presentation of the Augmented Transition Network (ATN) grammar formalism, see Woods (1970).

[4] Bobrow and Webber view this processing architecture in terms of Woods' (1980) Cascaded ATN model of interaction, although the exact relationship of the RUS/PSI-KLONE system to Woods' (1980, p9-11) syntactic/semantic cascade is uncertain.

in *Smith sent a message to Bill*) or the indirect object (as in *Smith sent Bill a message*). Subsequent syntactic information should resolve the ambiguity, but by considering the semantics of the NP in (12) PSI-KLONE may be able to dispose of one of the readings immediately. So if the fragment in question is

(13) Smith sent a message ...

PSI-KLONE can reject the syntactic reading in which *a message* is treated as the indirect object of *sent*, because it is generally implausible for a message to be sent anything.

How does PSI-KLONE make such decisions? Each message from the ATN to the semantic component takes the form of a triple

(M, L, C)

where M (for "matrix") represents the half-built constituent currently being parsed, C is a parsed and interpreted subconstituent, and L is the syntactic label which is proposed to link C to M. Thus, in parsing

(14) John ran the race

a message is sent after reading *ran*, proposing that *John* is the subject of the main clause. In this case, the (M, L, C) triple would contain the following information:[5]

(M = <a CLAUSE whose main verb is *ran*>,
L = SUBJECT,
C = <an ANIMATE NP>)

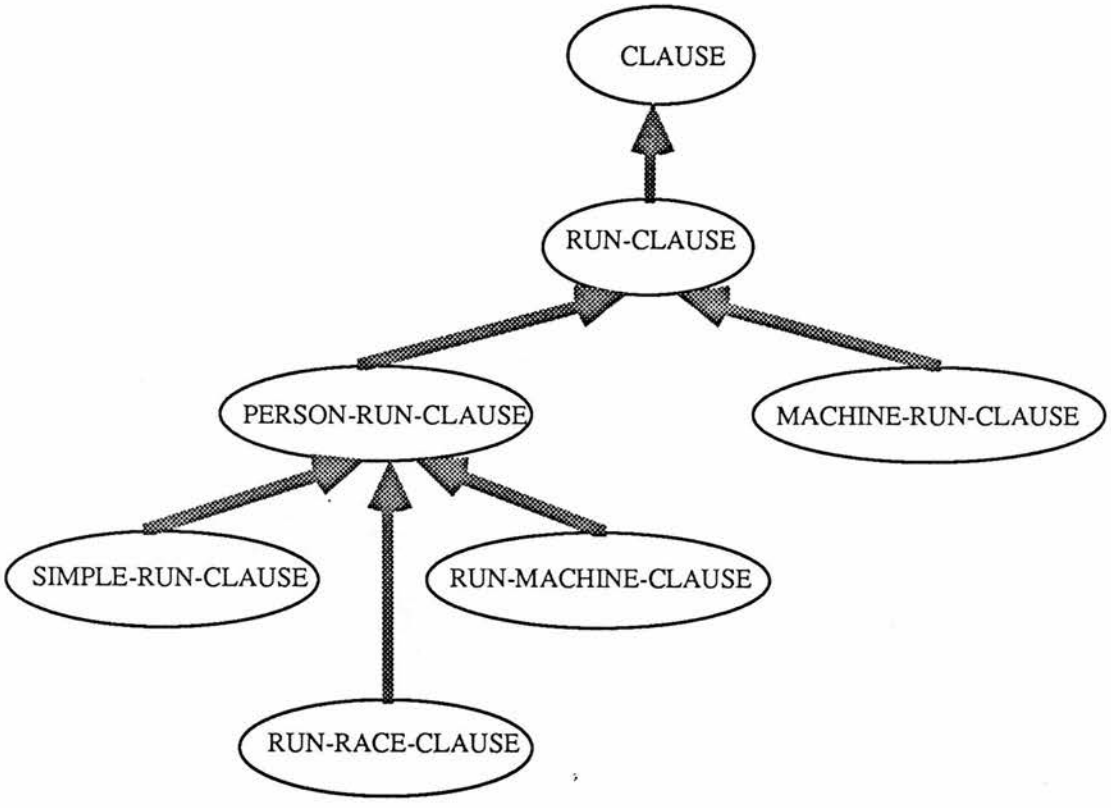
Note that the above triple includes both syntactic and semantic information, since *John* has already been interpreted by this same process of syntactic "proposals" at a lower level in the ATN. The semantic component assesses the general plausibility of the request by trying to relate the proposed syntactic label to a thematic role in one of its stored semantic case frames. For example, it might relate the above Subject label to the semantic role of Agent, since *John* is an Animate description and is in Subject position, and because one case frame for running requires an Animate-type Agent. The semantic component then updates the matrix structure M with this information and returns it to the parser, for use in later transmissions.

In essence, then, PSI-KLONE checks the semantic plausibility of a proposal — which contains both syntactic and semantic information — by determining its consistency with some semantic case frame. Interestingly, the system represents the set of syntactic/semantic proposals that can be interpreted (i.e. mapped into a case frame) in the intensional format

[5] As in all such examples throughout this chapter, the authors' original notation is simplified for

provided by KL-ONE, a formalism for representing generic knowledge (Brachman and Schmolze, 1985). This allows the set of syntactic/semantic shapes to be taxonomically organised in a lattice which specifies the way they share information.

By way of an example, Figure 2.1 illustrates the central links in a KL-ONE lattice specifying the connections between the various interpretable clauses about running.[6] In addition, each node in the lattice is associated with certain linguistic roles, and any node which it subsumes inherits these same roles in order to facilitate the structure-sharing. In



(Adapted from Bates, Bobrow and Webber, 1981, p107)

Figure 2.1. How recognisable *run* clauses relate to each other in KL-ONE.

the purposes of exposition.

[6] In fact the network in Figure 2.1 specifically constitutes a *tree*, rather than a *lattice*, but we will continue to use the latter term since in general KL-ONE permits a node to have multiple parents.

Figure 2.1, for example, the CLAUSE node would be linked to an obligatory Subject role, an obligatory Head role, and an optional Object role (these roles are not included in the figure because of their pictorial complexity). The taxonomy also indicates the types of linguistic object that may fill these roles; stating that the sentential Subject must be an NP, that the clausal Head should be a verb, and so on. Since the CLAUSE node subsumes the RUN-CLAUSE node, the RUN-CLAUSE node inherits all of these expected features. But, in addition, it expects its Head to be the verb *run*. So the RUN-CLAUSE node characterises a more specific shape than the CLAUSE node. In turn, RUN-CLAUSE subsumes PERSON-RUN-CLAUSE and MACHINE-RUN-CLAUSE, and each of these places further restrictions on the attributes of an interpretable running clause. Thus a sentence is recognised as a PERSON-RUN-CLAUSE only if its Subject NP is an Animate type of NP. In contrast, MACHINE-RUN-CLAUSE expects a machine-type NP as Subject.

PSI-KLONE's decision to accept or reject a syntactic proposal from the ATN depends simply on whether or not it can be located at some node in such a taxonomy. If the proposed syntactic/semantic shape satisfies the role restrictions of some node in the lattice, then it is assumed to be interpretable, and actions associated with the node can build the appropriate case frame when it is required. The important point is that the set of allowable syntactic/semantic shapes is represented intensionally, meaning that PSI-KLONE can incrementally develop the details of a particular clausal matrix by moving from node to node in the lattice as the ATN arcs are traversed.

By way of an example, consider the sentence in (15).

(15) John ran the drill press

In isolation, the sense of the verb *ran* is ambiguous: according to Figure 2.1, it may be relate to the running of a machine or the running of a person. But PSI-KLONE's taxonomic representation of linguistic sense allows the ambiguity to be resolved incrementally, as the sentence is read. After determining that *John* is an Animate type of NP, the parser reads the next word and proposes that *run* is the Head of a CLAUSE. The verb satisfies the restrictions which apply to the Head role of a RUN-CLAUSE and so creates an instance of this generic node. The parser can now make another proposal, to the effect that *John* should fill the Subject slot in the newly created instance of RUN-CLAUSE. The semantic interpreter accepts this, now linking the matrix to the PERSON-RUN-CLAUSE since the subject is animate. Finally the object NP *the drill press* is recognised. This distinguishes between the three specialisations of PERSON-RUN-CLAUSE, making the

sentence a RUN-MACHINE-CLAUSE. Since all the roles pertaining to this node are now filled, a complete semantic representation is built, on the basis of the type of description that has been recognised.

The point of interest in this example is in the way PSI-KLONE incrementally refines the sense of the verb *run* by gradually progressing down the inheritance lattice as information becomes available from the parser. This seems a very natural form of incremental processing. Intermediate results influence the interpretations of subsequent words and, correspondingly, each word makes the intermediate interpretation more specific. Importantly, later processing in the string cannot *change* earlier decisions; it can only make an earlier representation more specific. Here, then, incremental processing consists of gradually refining a set of possibilities by, at each stage, moving from a certain node in the lattice to one of the nodes which it subsumes.

Bobrow and Webber characterise their refinement mechanism as an instance of a general class of incremental process which they call Incremental Description Refinement:

> *Incremental Description Refinement* (IDR) is the process of:
>
> - Determining the set of descriptions compatible with an object known to have a given set of properties.
> - Refining the set of descriptions as more properties are learned.
>
> (From Bobrow and Webber, 1980b, p4)

In the domain of syntactic/semantic shapes, an "object" is some linguistic constituent, and the compatible descriptions are represented by nodes in the lattice. If PSI-KLONE's refinement operation has reached a given node in the lattice, the currently applicable set of compatible descriptions are just those terminal nodes subsumed by this node.

On these grounds, Bobrow and Webber's system constitutes an interesting example of an inherently incremental semantics. However, there are drawbacks for any design which excludes local referential knowledge from processing, as Bobrow and Webber do. For example, we have seen how sense-semantic knowledge can be effective in ruling out spurious readings of phrases like (16).

(16) The brick hit ...

Here, the analysis which treats *the brick* as Agent can be eliminated once the verb is read, since *hit* requires an animate-type Agent and *the brick* is not an animate NP. But there are well-known problems with attaching such selectional restrictions to verbs in this fashion, especially concerning "semantic neutral" terms such as those in (17).

(17) a. I am wearing my birthday present
b. I will drive my birthday present to Bristol

(Adapted from Bobrow and Webber, 1980b, p7)

PSI-KLONE would provide no reading at all for these sentences because the object NP *my birthday present* does not explicitly bear the features required by the verbs *wearing* and *drive*. For example, the sentence in (17b) would be judged anomalous because *my birthday present* does not explicitly bear the feature VEHICLE-NP, as required of any filler of the Objective role in the case frame for driving. Similar problems would arise with other neutral terms like *something*, elliptical forms like *the last two*, and certain pronouns.

To counter this problem, Bobrow and Webber (1980b, p7) suggest changing PSI-KLONE so that, instead of checking for features which guarantee consistency (such as the feature VEHICLE-NP as an object of driving), it checks that the features of a proposed filler are *not necessarily inconsistent* with the requirements of the given case frame slot. Thus *my birthday present* would be accepted as the object of *drive* because *my birthday present* is not necessarily a non-vehicle, and so is potentially compatible with being driven.

Not only does this move limit the force of any on-line semantic feedback to the parser, it will not extend to certain cases of metonymy, as Bobrow and Webber acknowledge. Natural language is full of expressions like

(18) The hamburger wants a large Coke

(Adapted from Bobrow and Webber, 1980b, p7)

such as might be uttered by a waiter in a restaurant; of course the waiter does not really *mean* a hamburger, but the person who ordered it. Unlike *my birthday present*, *the hamburger* is not semantically neutral, and will be treated as a positive instance of FOOD-NP. Since it is necessarily inconsistent for a FOOD-NP to have intentions, the sentence in (18) would be rejected as semantically anomalous.

However, there is an alternative approach to the problem of semantically neutral arguments that might also shed light on the difficulties with metonymy. McCawley (1968) suggests that selectional restrictions might apply to the *referents* of verbal arguments rather than their *senses*, if indeed they are to be applied at all. So for (17b) we retain the restriction that the object of *drive* should be a vehicle, but apply this restriction to the referent of *my birthday present* in the present context, rather than the sense of the expression. Furthermore, as Hobbs and Martin (1987) show, such a referential framework can accommodate NPs which are used metonymously, as in

(19) We disengaged the compressor after the lube oil alarm

where the NP *the lube oil alarm* can be interpreted as referring to *the sounding of* the lube oil alarm.

Treating referents as primary semantic objects would also bring us more into line with Crain and Steedman's hypotheses about the use of semantic information by the HSPM. They claim that if general knowledge about the world and specific knowledge about the discourse ever give conflicting advice on the plausibility of an analysis, the latter must take precedence over the former; a claim which is borne out by the evidence presented in A&S.

Our main criticism of PSI-KLONE, then, is that referential elements are excluded from its incremental semantics. There is, however, a second problem, concerning the incrementality of the sense-semantic process. The problem originates with the system's syntactic component (RUS), a parser for ATN grammars. Recall that as RUS processes a main clause with the form

NP_s V NP_o

(where the subscripts "s" and "o" are for subject and object, respectively) it builds up a syntactic/semantic representation of the clause as its constituents are found. RUS parses an NP in the same way: it enters an NP network, and as arcs are traversed it constructs a syntactic/semantic representation of the constituent being parsed (i.e. an NP). Once the NP has been parsed, its interpretation is sent up to the main clause level to be incorporated into the overall matrix structure. Unfortunately, this means that when processing an object NP (NP_o above), RUS must wait until NP_o is completely parsed before integrating it with prior semantic interpretations. This means, for example, that PSI-KLONE would not detect the anomaly in

(20) I chatted to the waste-paper basket that I left behind the door in the bedroom on the right upstairs

until the relativised object NP *the waste-paper basket that I ... upstairs* is closed and an attempt is made to treat its interpretation as the object of *chatted to*.

Note that this is a problem with incrementality rather than interaction. The syntactic and semantic components of RUS/PSI-KLONE do interact, more or less, on a word-by-word basis (or at least they have the power to do so). But because of RUS's hierarchically structured parsing process (suggested by the underlying ATN grammar), the semantics of a word is not always directly incorporated into the top-level semantic representation. In cases such as the above, then, RUS does not provide a truly incremental semantics; instead it provides isolated "pockets" of incremental interpretation, where the processing of an NP, in itself, is incremental, but where the NP is not linked to what has gone before until

it is syntactically complete.

2.2.2. Interactive Reference-Semantics

The above section argued that the main drawback with Bobrow and Webber's form of incremental semantics is that it does not utilise referential knowledge during syntactic processing. It therefore seems appropriate that we now turn our attention to those models which perform some form of reference evaluation on-line to a syntactic parser. Here we will consider the programs of Winograd and Hirst, which both demonstrate how the results of reference evaluation can interact with a syntactic processor to resolve certain structural ambiguities. However, as we will see, these proposals do not embody any relevant form of incrementation, and so in this sense they are less advanced than the proposals of Bobrow and Webber.

Winograd (1971, 1972, 1973) provides an early example of on-line referential processing in his SHRDLU program. SHRDLU, a simulated robot, operated in a simulated toy "blocks world", consisting of five blocks, three pyramids and one box. The robot could respond to simple questions and commands, as the sample dialogue below indicates (SHRDLU's responses are written in upper case).

(21) a. Pick up a big red block.
b. OK.
c. Grasp the pyramid.
d. I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
e. Is there a large block behind a pyramid?
f. YES, THREE OF THEM: A LARGE RED ONE, A LARGE GREEN CUBE AND THE BLUE ONE.
g. What does the box contain?
h. THE BLUE PYRAMID AND THE BLUE BLOCK.

(Adapted from Winograd, 1973, p155-161)

As we can see from this short illustration, SHRDLU's linguistic behaviour is very much wrapped up with the current state of the blocks world and its knowledge of the conversation — just as our own, human use of language is intimately related to the prevailing linguistic and non-linguistic context. For example, it cannot interpret the instruction in (21c) since it cannot establish a specific referent for the NP *the pyramid* — from the definite article *the* it expects the phrase to refer to a unique pyramid, and yet there are three in the blocks world. However, the structurally similar NP *the box* in (21g) does not cause any trouble, because SHRDLU knows about only one box.

Winograd achieves this sensitivity to the context by making the system's semantic processes dependent on the current state of the blocks world (and on its history of the prior dialogue). The blocks world is straightforwardly represented as a "database", or list of facts:

(22) (IS B1 BOX)
(IS B2 PYRAMID)
(IS B3 BLOCK)
(IS B4 BLOCK)

(COLOUR-OF B2 BLUE)
(COLOUR-OF B3 RED)
(COLOUR-OF B4 BLUE)

(CONTAIN B1 B2)

(Adapted from Winograd, 1973, p168)

The above fragment of knowledge expresses the fact that B1 is a box, that it contains a pyramid, B2, and so on.

SHRDLU interprets an input sentence by translating it into a program (in the problem-solving language Planner), and then executing the program with respect to its current set of blocks world facts. The question in (23a), for instance, might be translated into the procedure call of (23b):

(23) a. Does the box contain the red block?

b. (PROVE (CONTAIN B1 B3))

The system thus translates the Yes/No question into the goal of proving that the box, B1, contains the red block, B3; and its response to the user will depend on whether the proposition can be proved. It can carry out such proofs by checking that the database includes the fact (CONTAIN B1 B3), or perhaps by seeing whether B1 contains some other object which, in turn, contains B3. In a similar way, natural language commands are interpreted as calls to procedures which can execute actions in the blocks world. So the instruction *Pick up a big blue block* might translate as (GRASP B4) — an instruction which, when executed, simulates the appropriate robot action.

Notice that in procedure calls such as (23b), SHRDLU has already determined the referents of *the box* and *the red block*. In fact, referring expressions are evaluated in the same deductive manner as questions and commands. In the case of *the red block*, SHRDLU first assembles a procedure from the senses of the words in the NP:

(24) a. the red block

b. (FIND 1 ?X s.t. (PROVE (IS ?X BLOCK))
(PROVE (COLOUR-OF ?X RED)))

Here the variable ?X stands in place of the referent of the expression; the clauses (PROVE (IS ?X BLOCK)) and (PROVE (COLOUR-OF ?X RED)) stem from *block* and *red*, respectively; and the enclosing instruction to FIND 1 ?X is designed to capture the uniqueness intended by the use of the definite article. Once complete, this procedure is evaluated by searching in the database for all instances of ?X for which the twin properties (IS ?X BLOCK) and (COLOUR-OF ?X RED) are true. If just one entity fits the description (also considering information about the recent dialogue), then ?X is bound to this value; if no entities or more than one can be found, the procedure fails. So with the database of (22), the routine would succeed and bind ?X to B3.

Our present interest in Winograd's program concerns the way such reference evaluation is intermingled with syntactic processing. As the parser finds a complete NP in the string, it calls the semantic component to build an appropriate deductive procedure. This procedure is immediately evaluated to determine the contextually relevant referent for the NP. Moreover the success of this evaluation influences the direction taken by the parser: if a referent cannot be found (or if too many can be found), the parser tries an alternative syntactic reading of the string.

This strategy helps resolve ambiguities in the closure of definite noun phrases in sentences like (25):

(25) a. Put [the blue pyramid on the block]$_{NP}$ [in the box]$_{PP}$
b. Put [the blue pyramid]$_{NP}$ [on the block in the box]$_{PP}$

If the database shows there to be a blue pyramid on the block and, furthermore, this is uniquely identifiable either by virtue of being the only one or having been recently mentioned, then analysis (a) is adopted. However, if no referent can be found, analysis (b) is chosen — providing the blocks world contains a single blue pyramid and a single block in the box. It can be seen that this aspect of SHRDLU's context-sensitive behaviour constitutes an implementation of Crain and Steedman's (1985) later Principle of Referential Success, discussed above in section 2.1.

Let us now take a more critical look at Winograd's program, concentrating on two pertinent areas. First of all, there is little sense in which SHRDLU's semantic interpretation process can be regarded as "incremental". Rather than steadily refining a description (in the manner of PSI-KLONE, for example), the interpreter appears to operate in fits and

starts, interleaving specialised pieces of semantic processing with chunks of syntactic analysis. This problem is most acute with reference evaluation, since a referring expression is not evaluated at all until it is syntactically closed. Given that English grammar permits NPs to be arbitrarily long (as in *the man who said I didn't want to know that Harry had told* ...), this means that reference evaluation may be delayed indefinitely.

The lack of an incremental semantics relates to a second problem, concerning the frequency of interaction between syntax and semantics. SHRDLU's syntactic parser explores the syntactic search space depth-first, and in parsing an ambiguous sentence like (25) apparently first investigates the reading with the *longest* possible object NP.[7] Thus, in the case of (25), only once *the blue pyramid on the block* has been parsed as an NP will the mechanisms for semantic evaluation intervene and pass judgement on the analysis. However, as we noted when discussing Altmann's Principle of Referential Failure above, the attachment decision can be made earlier. It would be more conducive to the goal of incremental interpretation to first explore the reading with the *shortest* possible NP, and on the basis of this result decide where to attach the subsequent PP.

Hirst (1983a, 1983b, 1984, 1987, 1988) has more recently demonstrated how referential information can be brought to bear on syntactic ambiguity, following Winograd, and Crain and Steedman. Here we briefly consider his procedure for resolving attachment ambiguities involving prepositional phrases, as exemplified by

(26) The girl saw the boy with the telescope

Hirst's process of semantic interpretation is driven by an adapted version of Marcus' stack-and-buffer parser (Marcus, 1980), and during parsing this may call upon the developing semantic representations to help resolve an attachment ambiguity. Processing from left to right, the parser will first call for semantic guidance when it has just parsed a PP. At this point, the PP will occupy the first element of the parser's three-constituent buffer, and its possible attachment sites will be stored on the stack (Hirst considers the case where the PP may potentially attach to a single preceding VP or a number of preceding NPs). When asked for advice, the semantic component responds by forming semantic translations for the alternative analyses, evaluating them in a manner similar to Winograd, and analysing the results. So for the above example, the program would form two individual semantic translations, corresponding to the NP attachment and VP attachment of *with the telescope*. These are then evaluated, to see, for example, if one reading refers successfully (in the

[7] This observation is based on the example in Winograd (1973, p182).

sense of Crain and Steedman's Principle of Referential Success). The program then feeds the evaluation results through the decision algorithm in Figure 2.2, and makes the attachment accordingly. (The figure omits the details for choosing between analyses in terms of their general, context-independent plausibility and any verbal expectations.)

Note that Hirst's program resolves attachment ambiguities earlier in the string than SHRDLU since the algorithm of Figure 2.2 is invoked as soon as the shortest possible PP has been found, whereas SHRDLU initially looked for the longest possible complex NP. However, a number of problems remain. Like Winograd's program, Hirst's process of reference evaluation is not, in itself, incremental; rather it is something that happens, instantaneously and completely, at certain times during parsing. And these points of reference evaluation are not very early: in processing (26) above, no attempt would be made to evaluate the NP object of *saw* until the subsequent PP is parsed. Thus, like SHRDLU, interaction between syntax and reference evaluation is not as early as suggested by the Principle of Referential Failure.

So, from our point of view, the programs of Winograd and Hirst suffer from the same

---

If some NP attachment gives referential success
then attach to most recent such NP

else if {plausibility}

...

else if {verbal expectations}

...

else if there is an NP attachment known to be plausible that doesn't give referential failure
then make the rightmost such attachment

else if there is an NP attachment that doesn't give referential failure
then make the rightmost such attachment

else sentence is ambiguous, but prefer VP attachment anyway

(From Hirst, 1987, p174)

Figure 2.2. Hirst's decision algorithm for restrictive PP attachment.

---

problems. Although reference evaluation occurs during the parsing, the actual process of reference evaluation is not incremental. These programs do not incrementally resolve a reference as a string is read from left to right; rather, the syntactic closure of certain noun phrases triggers an immediate and complete process of evaluation. In addition the programs do not semantically test the shortest possible NP reading first of all, as the Principle of Referential Failure suggests they should do; this further delays the reference evaluation and unnecessarily postpones useful interaction with the semantic component.

2.2.3. Incremental Reference-Semantics

The work of Mellish (1981, 1982, 1983, 1985) is explicitly addressed to the first of our grievances with Winograd's and Hirst's programs, namely their lack of an incremental reference procedure. Mellish's approach is similar in spirit to Bobrow and Webber's work on incremental description refinement (IDR), though in the domain of reference evaluation rather than sense-semantic refinement. Mellish thus shares Bobrow and Webber's interest in intrinsically *incremental* semantic processes, and considers how referential information can be derived from a sentence as early as possible during a left-to-right syntactic analysis.

Why should Mellish want to provide an IDR process for reference evaluation? To set the scene, recall that Winograd's program evaluates the reference of a definite noun phrase when it has been read and parsed in its entirety. No reference evaluation is done before this point, and none is done afterwards; thus the phrase is expected to refer successfully once it is syntactically closed. Mellish questions this rigid view of reference evaluation, arguing that the time at which references are resolved in sentence processing is entirely pragmatic, and cannot be pinpointed in terms of the syntactic structure of the sentence or the sense of the referring expression. Instead he argues that:

> The notion of "referent" is something shaped by the context of a noun phrase's use, rather than a simple function of the phrase itself.
>
> (Mellish, 1982, p5)

Mellish claims that this phenomenon arises with both full and pronominal NPs, definite and indefinite. Consider the following sentences involving indefinite NPs, from Mellish (1985, p23-24):

(27) a. *Small blocks* are clamped at the ends and at the centre of a light rod
b. A bridge is supported by *a pier* at each end

Let us suppose that in (27a) the plural indefinite *small blocks* presents information new to

the discourse. Then, in (27a), it is clear that the total referential impact of this phrase cannot be entirely determined as soon as the phrase is syntactically analysed. *Small blocks*, in itself, introduces an indeterminate number of blocks into the discourse; inferences must be drawn from the rest of sentence to realise that three blocks have been introduced.

Mellish also points to examples where the reference of a definite NP might depend on subsequent material outside the NP, such as with temporal modifiers:

(28) a. *The President of the United States* signed the test-ban treaty *in 1962*
b. *The Tay Bridge* was blown down *in 1879*

(From Ritchie, 1976)

The claim here, following Ritchie (1976), is that the referent of the initial (italicised) NP cannot be determined locally once the NP is syntactically parsed. So in (28a), for instance, a processor must wait for the sentence-final verbal modifier *in 1962* before resolving the reference of *the President of the United States* to some particular President — in this case, John F. Kennedy.

On the basis of such evidence, Mellish concludes that there is no fixed time in sentence processing at which we can guarantee to be able to resolve the reference of an NP. He argues that the reference of an NP is dependent on the linguistic and non-linguistic context created by the enclosing sentence, previous sentences, and possibly even subsequent ones. Therefore, he argues, if reference resolution is to occur at all during syntactic analysis it *must* be done in an incremental fashion — in contrast to Winograd's program, which regards reference evaluation as a special-purpose activity centred around the completion of NPs. Mellish therefore sets about developing an inherently vague representation for references that can be incrementally refined as the text makes the disambiguating information available.

The work forms part of a natural language front-end to a large system designed to read in and solve elementary mechanics problems such as those found in school textbooks or examinations (the MECHO project; see Bundy *et al*, 1979). The input tended to consist of statements like

(29) Two pulleys of weights 12 lb and 8 lb are connected by a string passing over a fixed pulley

which, collectively, create a small mechanics "microworld" in which a problem is to be solved. This domain allows Mellish to concentrate on texts involving declarative sentences, and he can further assume that the discourse will introduce and refer to a small number of specific entities with well-understood physical properties. His program deals with a variety

of singular and plural referring expressions in this context. Although we will be concerned almost exclusively with the program's treatment of singular definite reference, it is worthwhile considering some of its general architectural properties first.

The overall process of semantic analysis is driven by a Definite Clause Grammar (DCG; see Pereira and Warren, 1980); a grammar, written as a Prolog program, with a direct procedural interpretation as a top-down parser, and similar in many respects to an ATN. In accordance with Prolog's control structure, syntactic ambiguity in the DCG is explored by a depth-first search, backtracking on failure in the familiar fashion. However, it is not clear how much structural or form-class ambiguity lies in the grammar, and Mellish's work is not directly addressed to the problem of syntactic disambiguation.

Semantic operations, built up from definitions in the lexicon, are invoked whenever the parser finds a major syntactic constituent. This leads to an order of semantic evaluation which Mellish (1985, p35) exemplifies with the following sentence

(30) A particle of mass b rests on the smooth table

in Figure 2.3. The type of semantic evaluation involved at each of the points in Figure 2.3 depends on whether the corresponding linguistic material is seen as providing *given* or *new* information (cf. our use of Halliday's terms in Section 2.1), in contrast to Winograd's program, which assumes all NPs refer to given, known entities. However, in order to avoid the immensely complex issue of determining the informational status of a fragment of sentence, the program assumes all definite NPs are given, and all indefinite NPs and verbal constituents at the main clause level are new. So in the sentence in (30), the indefinite *a particle of mass b* is taken to introduce a new, specific particle into the context; this is modelled by creating an instance of a particle, giving it a name, say PARTICLE1, and adding this to the program's database. The definite NP *the smooth table*, on the other hand, is assumed to refer to some table already present in the context; and in this case the

---

A *particle*	*of mass*	*b*	*rests on*	the	*smooth*	*table*
(1)	(2)	(3)	(6)		(5)	(4)

(From Mellish, 1985, p35)

Figure 2.3. Order of semantic evaluation by Mellish's program.

---

program starts to resolve the reference by collecting together a set of candidate entities from its database.

How are definite references resolved by the program? Mellish sees semantic interpretation as a process of satisfying the "constraints" which must hold if a sentence is to make sense, and reference resolution simply comes about as a by-product of this more global procedure of constraint satisfaction. These constraints — which may come from the general context, the sentence, and the noun phrase itself — gradually accumulate as the sentence is parsed; and as the constraints are imposed, the number of candidate referents which satisfy them shrinks.

The program can derive two sorts of constraint from a text, and these are both useful in evaluating references. In evaluating the reference of *the smooth table* in (30), for example, it derives the constraints

smooth(X) & table(X)

from the NP, meaning that whatever the referent X is, it must be smooth and it must be a table. These constraints express the defining characteristics of a smooth table; they can be applied to entities in the database to pick out just those objects that fit the description. (The above constraints are of the same variety as Winograd's (PROVE (IS ?X BLOCK)) deductive procedure calls.) Whereas these constraints arise from *given* information, a second source of constraint can be derived from both the *new* and *given* information in the sentence. This concerns knowledge about the physical possibilities of the world, which can act as a "semantic check" on all information expressed in the sentence. For example, the adjective *smooth* also imposes the constraint

isa(surface,X)

meaning that the referent X must be of type "surface"; in other words, anything that is described as being smooth must, in principle, be a surface, irrespective of whether it is new or given in the discourse. Another general physical constraint stems from the verb group in (30): the referent X must be something which, in principle, can support a particle. Thus X will also be subject to the constraint

can_support(particle,X)

— a precondition for the meaningfulness of the sentence as a whole.

Given these sources of constraints, Mellish formalises the actual process of reference resolution in terms of constraint satisfaction, in the spirit of Waltz' (1975) celebrated "filtering" algorithm (see also Mackworth, 1977a). As Mellish demonstrates, we can do this by representing a partially-evaluated reference by a symbol, such as ref(3). And with

each such symbol we associate a candidate set of potential referents from the context. To begin with, this candidate set will contain all those discourse entities consistent with the initial constraints on the reference. But as constraints on the reference accumulate, the candidate entities no longer consistent with the constraints are eliminated or "filtered" out of the set.

The candidate set may, eventually, reduce to a single element, in which case the reference has been fully resolved. Or the imposition of some constraint may actually eliminate every entity from a certain candidate set, meaning that the syntactic analysis which derived the constraints is inconsistent with the program's view of the world. In such cases the parser is forced to abandon the analysis and backtrack to a previous choice point, retracting constraints as it goes.

Formal constraint satisfaction — or "network consistency", to use Mackworth's (1977a) more accurate terminology — seems a natural form of incremental description refinement, providing a straightforward representation for partially-evaluated references in terms of candidate sets. It also offers a means of resolving related references in parallel, by the technique of constraint propagation. Whenever a discourse entity is eliminated from a particular candidate set, the effects are "propagated" to any related references in case their candidate sets have to be revised too.

To appreciate the workings of Mellish's reference procedure, consider the example

(31) A uniform rod is supported by a string attached to *its* ends
(From Mellish, 1985, p49-52)

and the problem of determining that *it* refers to the rod, and not the string. Supposing that this sentence occurs at the beginning of a text, by the time the program reads *its* it will be aware of two entities: a uniform rod, ROD1, and a string, STRING1.[8] Symbolising the reference of *it* as ref(1), the program therefore creates an initial candidate set consisting of just these elements, {ROD1, STRING1}, and associates this with ref(1).

The next step is to apply the meaning of *ends* to ref(1). First of all, this entails checking that both candidates for ref(1) can, in principle, have ends. They both pass the test, and the program now starts to resolve the reference of *ends* by imposing the following constraints:

---

[8] From Mellish (1985, p49), when describing this example. Since these entities have been introduced by the time *its* is read, and we know such semantic operations coincide with the recognition of major constituents, it seems likely that *a string* is parsed as an NP even though it is part of a larger, modified NP *a string attached to its ends*.

leftend(ref(1),ref(2))
rightend(ref(1),ref(3))

These constraints introduce two more undetermined references — the left-end, ref(2), and the right-end, ref(3), of whatever ref(1) is. Because the system can infer that ROD1 and STRING1 both have left- and right-ends, ref(2) and ref(3) initially each have two candidates. At this point, then,

ref(1) is one of {ROD1, STRING1}
ref(2) is one of {ROD1-leftend, STRING1-leftend}
ref(3) is one of {ROD1-rightend, STRING1-rightend}

The program now turns to the new information that STRING1 is *attached to* the ends of ref(1), and so begins to assert the appropriate new propositions in the database. One precondition relating to such assertions is designed to reflect the fact that it is pragmatically odd to speak of a physical object being attached to its *own* ends. Thus as each attachment proposition is added to the database, a precondition of separability is checked. So, first, the program imposes the preconditional constraint:

separable(STRING1, ref(2))

In attempting to satisfy this, the program sequentially instantiates ref(2) with the two potential referents in its candidate set. The constraint holds where ref(2) = ROD1-leftend but not where ref(2) = STRING1-leftend, because an object is not considered separable from itself. The string's left-end is therefore removed from the candidate set of ref(2), meaning that now

ref(2) is one of {ROD1-leftend}

The effects of this refinement must now be propagated to any other unresolved references related to ref(2). Given that ref(2) is related to ref(1) by the leftend constraint, the program must ensure that all candidates for ref(1) remain consistent with the filtered set for ref(2). At this point STRING1 is eliminated as a candidate for ref(1), since it does not have a left-end in the set for ref(2). So now,

ref(1) is one of {ROD1}

Since ref(1) has changed, ref(3) must be reconsidered. This, in turn, resolves the reference of the right-end:

ref(3) is one of {ROD1-rightend}

Hence, once the second precondition of separability,

separable(STRING1, ref(3))

is imposed, it is trivially satisfied. The program maintains consistency in this manner as it parses the remainder of the sentence; ROD1 will satisfy these further constraints and so

will be accepted as the referent of *it*.

Of the work we have reviewed, Mellish's program clearly comes closest to meeting the objectives of this thesis. Mellish shows how reference resolution can be performed incrementally, on-line to a syntactic processor. In particular, Mellish has identified network consistency among constraints as a suitable IDR process for reference evaluation. In this thesis we will follow Mellish in employing a network consistency process for incremental reference evaluation, and we explicitly adopt the general class of such algorithms as a principal starting point for the present work in Chapter 4.

Nevertheless, there are some shortcomings with Mellish's system. The main drawback concerns the degree to which the constraint-based process of semantic interpretation directly follows the left-to-right scan of a sentence. As we can see from the program's order of semantic evaluation exemplified in Figure 2.3 above, the semantics is not truly incremental in a number of respects. First of all, verbs are processed after their arguments, meaning that a noun phrase subject is only combined with an adjacent right-hand verb group once all other arguments to the right of the verb have been found. This strategy contradicts our intuitions about incremental interpretation and predicts, for example, that the semantic disagreement of subject and main verb in (32), where the verb *believes* expects an animate subject, is not noticed until reading the full-stop.

(32) # $[\text{A fixed pulley}]_{NP}$ $[\text{believes}]_{V}$ $[\text{the two small particles.}]_{NP}$

Only at the end of the sentence will the parser attempt to combine the representation for the verb with those for its subject and object NPs, and so discover the inconsistency. This policy of processing functions after their arguments naturally means that in complex noun phrases like

(33) a. the men who ran the race
b. the men in the race

the simple NPs *the men* and *the race* are individually evaluated before any combination with the verb or preposition. In addition, prenominal, adjectival modifiers are processed after the noun they modify.

Thus, from our perspective, the main problem with Mellish's system is that it fails to implement a proper left-to-right incremental semantics; when a verb, preposition or adjective is read, its semantic interpretation is not immediately incorporated into the interpretation of the preceding fragment of sentence. Although Mellish (1985, p32) acknowledges this point, it is clearly not a problem which seriously hampers his account of the data he addresses (of which (27) is representative). Mellish's main objective is to show how the

reference of a noun phrase can be incrementally evaluated as the enclosing sentence is interpreted, just as Bobrow and Webber show how various sense-semantic properties of a subject noun phrase may be gradually established as the remainder of the sentence is parsed. And he achieves this objective (for definite NPs) by basing the process of semantic interpretation on network consistency; a computational procedure which provides incremental refinement at the appropriate level of description. It is of less importance that the individual steps of the semantic interpreter are not in a one-to-one correspondence with the steps of the processor which reads the input string from left to right.

However, our goal is different. We are more interested in an actual left-to-right evaluation procedure, and especially in the incremental evaluation of a referring expression *as that referring expression is parsed*, rather than as the enclosing sentence is parsed. In fact, this thesis will question Mellish's and Ritchie's view that the reference of a (given) full definite noun phrase may depend on subsequent sentential information external to that noun phrase (cf. example (28)). We will return to this point in detail in Chapter 7, where Mellish's approach to definite reference is contrasted with our own.

It is worth making two further remarks about the scope of Mellish's work. Firstly, it does not explicitly address the problem of syntactic ambiguity resolution. No doubt the system's DCG *does* contain ambiguities which are resolvable on semantic grounds, but from the written descriptions of the program it is difficult to tell where in the syntax these ambiguities lie and in which order the parser would explore the alternatives. Secondly, as Mellish (1985, p114) himself observes, the interface between syntactic and semantic processes "lacked precision". The present work will aim for a more explicit connection between syntactic and semantic structures and processes, in order to make the resultant model easier to assess and extend.

2.3. Conclusion

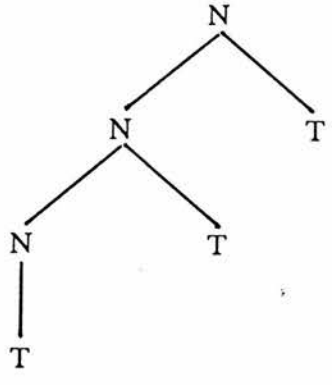
The computational accounts of on-line semantic interpretation we have discussed each place emphasis on different aspects of the comprehension process and, accordingly, they are deficient in alternative respects. But from the position of this thesis, they all share one common problem: to a greater or lesser extent, the models lack a truly left-to-right, incremental process of semantic interpretation. Given that the present thesis directly confronts this issue, it is illuminating to consider why these models suffer from such a problem.

We can begin to answer this question by observing that in all the above programs the

procedures for semantic interpretation are, in the main, *syntactically driven*. In each case, overall control tends to lie with the parser, which is responsible for finding syntactic constituents in the input string. Once a syntactic constituent is identified, its semantic interpretation is computed and stored with the syntactic category or structure representing that constituent. More specifically, the parsing operation of forming a syntactic constituent from syntactic subconstituents is used to trigger an isomorphic process at the semantic level, which produces a semantic interpretation for the syntactic constituent directly from the interpretations of its syntactic subconstituents. Thus the systems we have considered appear to adopt, in some sense, the *rule-to-rule hypothesis* formalised in Montague (1973); and so assume that the rules of syntax are in a one-to-one correspondence with the rules of semantics.

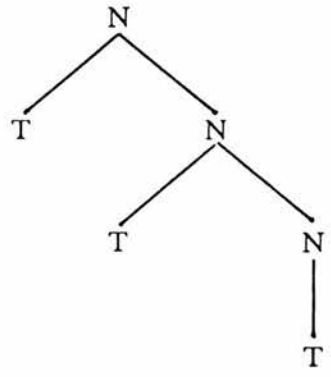
If we do indeed assume some variety of rule-to-rule compositional semantics, and we further assume that the process of semantic interpretation is driven by the syntax rather than the semantics, then the capacity of a given system for *incremental* interpretation is simply a question of grammar. For example, a grammar which produces left-branching phrase-structure trees, of the form in (34), should be most conducive to incremental semantics. (In (34), N represents a non-terminal category whereas T represents a terminal category.)

(34)



If an input sentence can be analysed as (34) then, in a left to right pass through the string, a new syntactic constituent N can be formed when each word is read. In accordance with the rule-to-rule assumption, this means that a semantic interpretation can be provided for each left-hand substring of the sentence, thereby effecting an incremental interpretation. In contrast, with a grammar which produces right-branching structures of the form in (35),

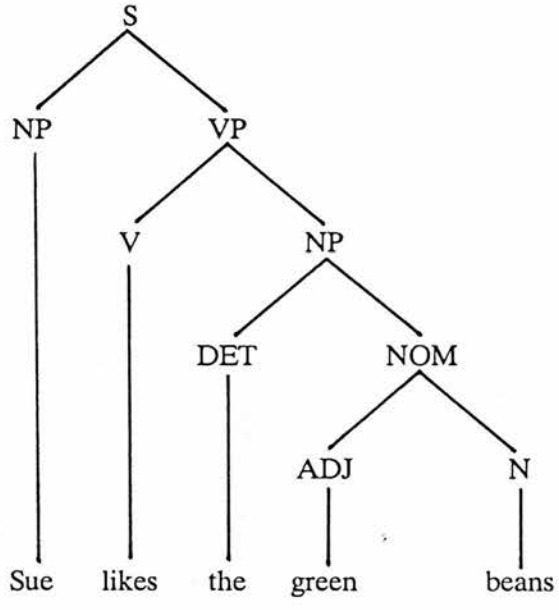
(35)



the rule-to-rule hypothesis implies that semantic interpretation must be delayed until the end of the sentence is reached. At this point, a processor can recurse back through the sentence, building syntactic and semantic representations from right to left.

But of course English is widely held to be a principally right-branching language, dominated by structures such as (36):

(36)



And since the computational models we have considered all employ more or less standard theories of grammar, it is not surprising that they fail to implement a properly left-to-right incremental semantics.[9]

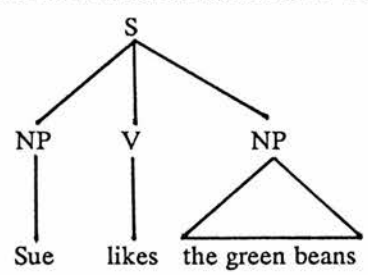
This thesis will retain the assumption that the process of forming syntactic and semantic constituents should proceed in a compositional, rule-to-rule fashion. And we will also continue to assume that the process of semantic interpretation is driven by syntactic rules. But we will differ from previous approaches in assuming a theory of grammar that does not yield conventional surface structures for English. As the next chapter explains, Steedman's

extended version of Categorial Grammar provides *left*-branching analyses for many constructions previously held to be *right*-branching, so making it directly compatible with the incremental style of interpretation we seek to implement.

---

[9] We should note that Bobrow and Webber's system manages to by-pass this problem to a certain extent, by flattening the structure of main clauses such as (36). So the syntactic structure implicit in their ATN does not include a VP node, and instead implies the following analysis of (36):

(37)



The procedural, non-compositional aspects of RUS's ATN are then exploited to to check the consistency of the concept *Sue likes*, even though it is not recognised as a syntactic constituent.

Chapter 3

Combinatory Categorial Grammar

Two principal foundation stones underpin our system of incremental interpretation, the core of which is presented in Chapter 5. The first supports pragmatic processing, providing for the incremental construction of referential descriptions derived from the context. Here our foundation comes from work on network consistency techniques in artificial intelligence, and we will consider this in detail in Chapter 4.

The second theoretical basis is syntactic, and for this we import an extended version of Categorial Grammar. The advantage of Steedman's Combinatory Categorial Grammar is that it allows incremental syntactic analysis, and so provides a convenient hook on which to hang the process of semantic interpretation. By ensuring that this process is directed by an incremental syntax, we are automatically ensured of an incremental semantics. The purpose of the present chapter is to review Steedman's theory, its associated parsing methodology, and its implications for incremental processing.

3.1. Extending Categorial Grammar

Categorial Grammars, originally conceived by Ajdukiewicz (1935), consist of two components. The first is a lexicon, associating each word in the language with at least one syntactic category. These categorisations fall into two camps: certain items, like verbs, are assigned functions, while others are seen as arguments. The second component of a categorial grammar is a set of rules for combining function and argument categories to produce new categories. Ajdukiewicz' original set contained just one rule, that of function application. However, in a recent series of articles, Ades and Steedman (1982) and Steedman (1985, 1987b, 1988, in press) extend the combinatory component with further category-combining rules. Foremost among these is the rule of function composition. Given the close relationship of these extensions to the Combinatory Logic,[1] the new, extended grammar has been dubbed *Combinatory Categorial Grammar* (CCG).

Steedman's extensions to traditional categorial grammar are motivated by purely linguistic concerns, such as the need to account for constructions involving unbounded

extraction and non-constituent coordination. But these additional rules have other attractions, and our especial interest is in the suggestion that the rule of function composition should be useful for incremental processing (Steedman, 1987a). The following sections therefore briefly motivate the relevant aspects of the theory from a linguistic standpoint, and go on to demonstrate its usefulness for incremental interpretation. Of the various versions of CCG which have appeared, the one presented in Steedman (1987b) will be assumed for the purposes of this thesis and so also forms the basis for our review of the theory. If at any point in need of further linguistic detail, the reader is referred to the primary source.

3.2. The Categorial Lexicon

The first component of CCG is a lexicon linking words to categories. The lexicon associates arguments, such as nouns, with atomic categories. So an entry for the noun *cake* will relate it to the atomic category N. Functions, on the other hand, are written as complex categories indicating syntactic mappings. For example, determiners are seen as functions mapping Ns to NPs. So the definite article *the* is defined as an NP/N, meaning that it combines with an N on its right to form an NP.

Some functions expect their arguments on the left instead. Whereas we use a rightward slash "/" when the argument is expected on the right, we use a leftward slash "\" to indicate the opposite. So the tensed intransitive verb *smiled* translates as the category S\NP, meaning that if this is given a (subject) NP argument on the left, then it will form a sentence.

There are no restrictions on the *types* of argument to a function; indeed, one function may become the argument to another. In principle, functions may also take an arbitrary *number* of arguments. The only restriction is that functions which take more than one argument are "curried", so that they take their arguments one at a time. This makes all functions unary: every function specifies a single mapping from an argument to a result, though the result may itself be another function. Thus the tensed transitive verb *ate* is categorised as (S\NP)/NP: a function from an object NP on its right to another function, which, when given a subject NP on its left, yields a sentence.

---

[1] See Steedman (1988) and references therein.

3.3. Function Application

The other main component of CCG consists of a set of combinatory rules, and the first of these, function application, is common to all categorial grammars. Function application allows a function to combine with an adjacent argument, so long as it is of the appropriate type. There are two instances of this rule in CCG: one for the case where the function precedes its argument in the string (*forward application*), and the other for where the argument comes first (*backward application*). In both cases, the function and argument must be *adjacent* in the string.

Adjacency is an important property, and one which holds for all combinatory rules in the grammar. The rules can only combine constituents that are next to each other. Another universal feature of the rules regards the link with semantics. Each rule defines a semantic operation to parallel the syntax, in a manner akin to Montague's (1973) rule-to-rule hypothesis. Consider, for instance, the rule of forward application:

(1) Forward Application (">a")

X/Y : f Y : y ==> X : (f y)

(The symbol ">a" will appear in subsequent derivations to signify the use of this rule.) Given a constituent of category X/Y and an adjacent right-hand argument of category Y, the rule of function application derives a category X. Here X and Y are variables ranging over syntactic categories. Semantics is separated from syntax by a colon and, as its name suggests, the semantics of this rule is to apply the semantic function f to the semantic argument y; in the terms of the lambda calculus this is indicated by the juxtaposition (f y).

The rule of backward application is similar in form to (1), except the order of function and argument is reversed:

(2) Backward Application ("<a")

Y : y X\Y : f ==> X : (f y)

Notice that the semantics of (2) is still function application, as the only difference between (2) and (1) is constituent order.

In the theory of combinatory grammar such lambda-expressions provide a structural blueprint for the natural language semantics, rather than providing the details of the semantics itself.[2] For the purposes of exposition we will follow this lead, and illustrate the rules with semantic translations also set in the schematic terms of the lambda calculus. However, these translations are purely fictional and are invented solely for purposes of

[2] Note that Steedman expresses the semantics using combinators like **B** and **S**, but we will

exposition; in Chapter 5 we will adopt a rather different stance on the issue.

Let us now turn to a concrete example. Suppose the lexicon defines (":=") the proper noun *John* and the tensed verb *smiled* as follows:

(3) John := NP : john'
smiled := S\NP : λx. smiled'(x)

We can use backward application to derive the syntax and semantics of *John smiled* in the following way:

```
(4)     John              smiled
        --------          ------------------
        NP:john'          S\NP:λx.smiled'(x)
        ---------------------------------<a
            S : (λx.smiled'(x)  john')
```

By beta-conversion, the above function application

(λx.smiled'(x) john')

converts to

smiled'(john')

the semantic translation of the sentence.

This example not only illustrates the workings of function application, but also shows that it is unnecessary to build a phrase-structure tree for a sentence to determine its semantics — a point which dates back to Woods' (1973) ATN. Instead the syntactic parts of the combinatory rules direct the immediate construction of the semantic translation. In CCG syntax is seen as a means to an end, rather than the end itself; syntax is something that is "done" rather than built explicitly.

3.4. Function Composition

Consider the categories in the following topicalised sentence:

```
(5)  Those apples,  Harry      must       eat
     ------------   -----      ---------  -----
          NP          NP       (S\NP)/VP  VP/NP
```

(Here, and in other examples, the atomic category VP is used as an abbreviation for an infinitival S\NP function.) (5) involves a discontiguous dependency in which the object NP has been extracted to the left, and presents a problem for a grammar employing just function application. Function application is of no direct use, given the stipulation that it

---

continue to use the equivalent but more readable expressions in the lambda calculus.

relates only *adjacent* items in the string.

In order to account for examples like (5), CCG imports the rule of function composition to combine *functions* that are adjacent. Like function application, this rule has a number of instances. The predominant one is forward composition:

(6) Forward Composition (">c")

X/Y : f Y/Z : g ==> X/Z : λz. [f (g z)]

(In rules of composition like (6), we will sometimes refer to X/Y as the *principal functor* and Y/Z as the *subsidiary functor*.) Naturally enough, the semantics of the rule is to compose the corresponding lambda-functions f and g.

Returning to (5), function composition allows *must* and *eat* to combine as follows:

```
Those apples,   Harry      must        eat
------------    -----    ---------   -----
     NP           NP     (S\NP)/VP   VP/NP
                         ----------------->c
                            (S\NP)/NP
```

The significance of this move is that it has brought the function in the sentence, (S\NP)/NP, adjacent to its subject and object arguments. (We will address the remaining problem, of function and argument order, in a moment.)

In addition to the above, the following instance of composition will be helpful for the syntax of right-extraposition and other constructions. In this rule the directions of the slashes "cross":

(7) Crossed Backward Composition ("<xc")

Y/Z : g X\Y : f ==> X/Z : λz. [f (g z)]

Other examples demand that the forward rule of (6) be generalised, so that the Y category can be arbitrarily embedded in the "result" part of the subsidiary functor:

(8) Generalised Forward Composition (">c")

X/Y : f (Y/W)/Z : g ==> (X/W)/Z : λz.λw. [f (g z w)]

In fact, this is only one of a set of possible *generalised* composition rules: here the Y category is embedded to just one degree, but in principle it may be arbitrarily embedded. Generalised composition allows the following combination of auxiliary and ditransitive verb:

```
(9)    must          give
       ---------     ----------
       (S\NP)/VP     (VP/NP)/NP
       ----------------------->c
            ((S\NP)/NP)/NP
```

3.5. Type-Raising

The lexical and combinatory units of CCG are supplemented by a set of type-raising rules for rewriting argument categories as function categories. These rules fall outside the grammar's combinatory component since they do not *combine* categories; instead they operate on individual categories, optionally rewriting them into their alternative "type-raised" forms.

Consider the stage we have reached in deriving the sentence in (5). The operation of type-raising will allow us to to combine *Harry* with *must eat*, by changing the NP's syntactic status as an argument to that of a function. On the one hand, *Harry* is an NP and, as such, may be the argument to a number of functions, including an S\NP to its right. But on the other hand, we can view *Harry* as a function over those functions which may apply to NPs. Functionally, then, *Harry* may also be seen as an S/(S\NP): given a predicate S\NP, it will yield a sentence. Thus one rule which "raises the type" of a constituent is the following, useful when applied to subject NPs:

(10) Subject Type-Raising ("ts")

NP : x ↑ S/(S\NP) : λf.(f x)

The semantics of this rule is exemplified later. Syntactically, it takes us one step further in our problematic derivation:

```
Those apples,  Harry        must    eat
------------   -----        ----------
     NP          NP          (S\NP)/NP
               --------ts
               S/(S\NP)
               ------------------------>c
                        S/NP
```

By raising the syntactic type of *Harry*, it may compose with the subsequent (S\NP)/NP to produce a single S/NP function, now adjacent to its extracted argument.

Unfortunately the direction of the slash in this S/NP function does not permit a further combination with its object argument to the left. But we are clearly very close to an analysis of the whole sentence, and to complete it we may invoke another type-raising

rule, this time useful for topicalised NPs.

(11) Topic Type-Raising ("↑t")

$NP : x \quad \uparrow \quad S/(S/NP) : \lambda f.(f\ x)$

The rule in (11) allows any NP to be turned into a function which, when given a sentence lacking an object, produces a complete sentence. So the above derivation proceeds by raising the type of the topicalised NP *those apples* and applying the resultant function to S/NP, producing an S:

```
Those apples,   Harry    must     eat
------------    ------------------
     NP                 S/NP
  --------↑t
  S/(S/NP)
  ------------------------->a
              S
```

In this thesis we will be more concerned with relativisation than topicalisation, although the approach is much the same for both constructions. Object relative pronouns, such as *which* in *apples which Harry eats*, are categorised as functions from objectless sentences S/NP into noun modifiers N\N. So an object relative clause is analysed as follows:

```
(12)  (apples)      which        Harry        eats
                  ------------   -----     ---------
                  (N\N)/(S/NP)    NP       (S\NP)/NP
                                 --------↑s
                                 S/(S\NP)
                                 ------------------->c
                                          S/NP
                  ------------------------->a
                               N\N
```

Subject relative pronouns are similar functions, except their domain consists of sentences missing their subjects. Giving them the category (N\N)/(S\NP) allows the following derivation:

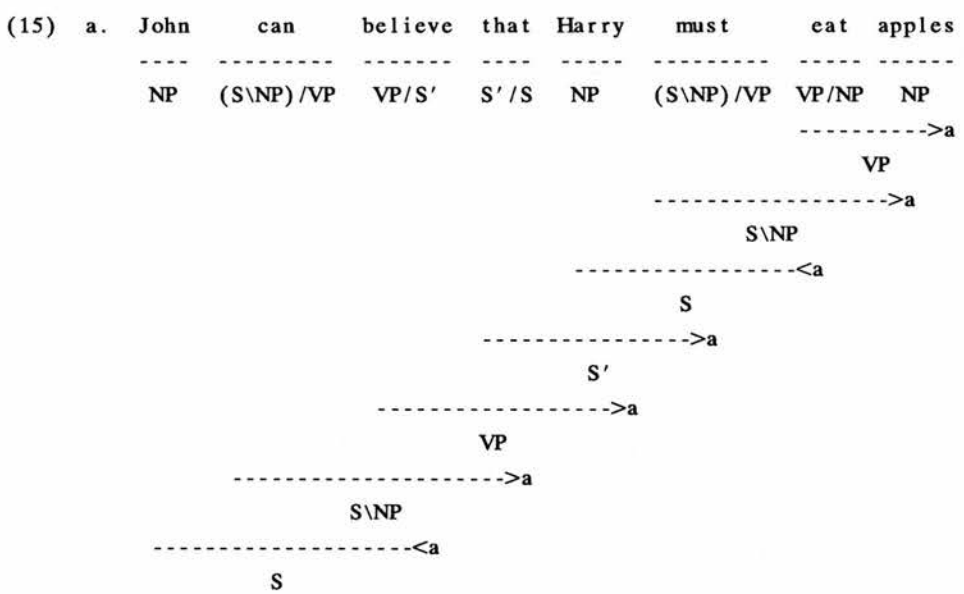
```
(13)  (girl)      who          smiled
                ------------   ------
                (N\N)/(S\NP)    S\NP
                ------------------->a
                         N\N
```

3.6. Composition, Type-Raising, and Incremental Processing

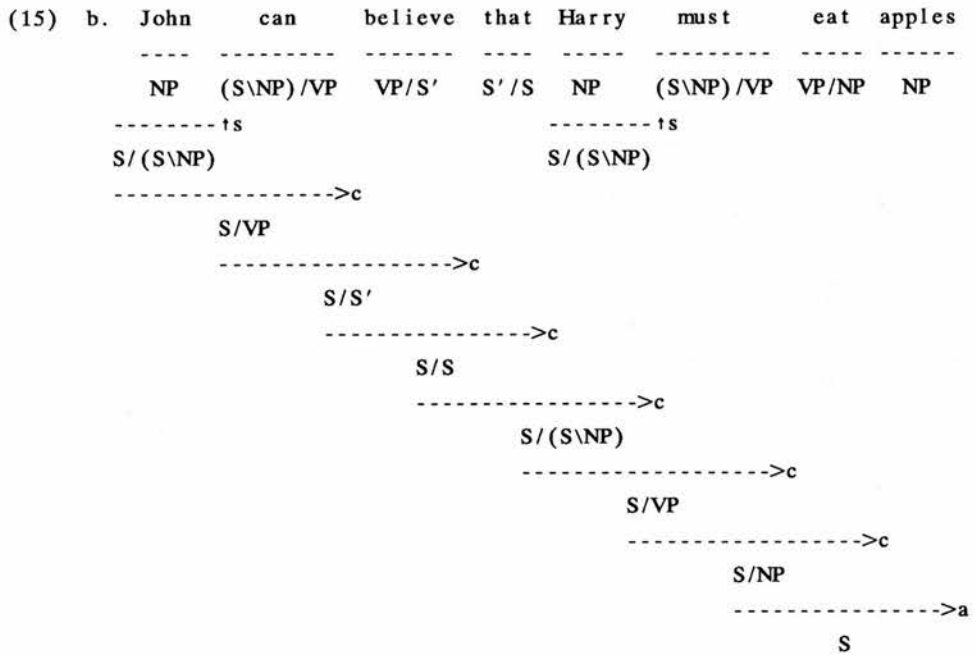
The above sections should give some indication of how Combinatory Grammar approaches the problem of unbounded dependency: function composition and type-raising work together to build a bridge between the function and its extracted argument. These aspects of the grammar have other attractions too, and in particular make the theory conducive to incremental processing. The mechanisms of composition and type-raising do not have to be tethered to those phrases whose analyses *require* more than just function application; they can be employed freely to provide incremental analyses of canonical sentences involving no movement at all. For example, the sentence

(14) John can believe that Harry must eat apples

can be analysed by function application alone, implying a right-branching syntactic structure:



But the availability of function composition and type-raising permits an alternative, incremental analysis of (14):



The analysis in (15b), being left-branching, gives syntactic status to every prefix of the string *John can believe that Harry must eat apples*. The grammar thus takes a radical stand on the issue of surface structure, predicting that fragments such as *John can believe that Harry must* are in fact constituents of the language. Indeed, the grammar allows a number of analyses of the unambiguous sentence in (14), corresponding to the different orders in which categories can be composed and applied (and each implying a different mixture of left- and right-branching in the associated phrase-structure tree). However, despite the syntactic differences, these alternative derivations are all *semantically equivalent*.[3] The rules of type-raising and function composition produce exactly the same semantic results as the more conventional derivation, which combines from the right and relies on function application alone. We can therefore exploit their potential for incremental processing without prejudicing the final semantic outcome.

To appreciate this point, let us work through the semantics of two alternative derivations of (16).

(16) John eats apples

One analysis involves just function application:

[3] Given certain assumptions about the nature of the semantic interpretations; see Steedman (1987b, pp. 416-417) for a brief discussion.

```
(17)  a.   John        eats        apples
           ----        ---------   ------
            NP       (S\NP)/NP       NP
                     ---------------->a (i)
                           S\NP
                    ----------------<a (ii)
                            S
```

Supposing that *eats* translates as λy.λx.eats'(x,y), the first combination in (17a) makes the application

(i) *eats apples*

(λy.λx.eats'(x,y) apples') ≡ λx.eats'(x,apples')

The second combination fills in the "subject" slot of the predicate, by application and beta-conversion:

(ii) *John eats apples*

(λx.eats'(x,apples') john') ≡ eats'(john',apples')

A second analysis involves type-raising and composition:

```
(17)  b.   John          eats       apples
           ----          ---------  ------
            NP         (S\NP)/NP       NP
          --------ts (i)
          S/(S\NP)
          ---------------------->c (ii)
                    S/NP
                    ------------------->a (iii)
                              S
```

The first step is to type-raise *John*, producing an abstraction in accordance with the semantics of the NP and the type-raising semantics of rule (10):

(i) *John*

λf.(f john')

This function is then composed with that representing *eats*:

(ii) *John eats*

λz.[λf.(f john') (λy.λx.eats'(x,y) z)]
≡ λz.[λf.(f john') (λx.eats'(x,z))]
≡ λz.[λx.eats'(x,z) john']
≡ λz.eats'(john',z)

And a final application binds z:

(iii) *John eats apples*

(λz.eats'(john',z) apples') ≡ eats'(john',apples')

Thus both analyses (17a) and (17b) produce the same semantic translation. This result follows partly from the fact that function composition is an *associative* operation and preserves the underlying semantic structure based on function application; note that the semantics of function composition (see (6)) simply involves an abstraction into a nesting of function *applications*. The semantic equivalence of the two analyses is also due to type-raising; in the syntax, an argument becomes a function, but in the semantics it remains an argument. We can therefore exploit the benefits which function composition and type-raising bring for incremental processing without affecting the semantics.[4]

The usefulness of rules like these for incremental interpretation has also been demonstrated by Hinrichs and Polanyi (1986) and Pulman (1986). Hinrichs and Polanyi directly import a version of CCG to help produce partial semantic interpretations for syntactically incomplete utterances, as found in ordinary interactive discourse. Pulman augments a conventional phrase-structure grammar with a processing operation which discards syntactic information from time to time during parsing (called "Clear"), using this to model memory limitations in human sentence processing. However, the semantic component of this operation corresponds to function composition, which, as Pulman shows, makes his model capable of incremental interpretation.

3.7. Type-Raising and Incremental Processing: A Closer Look

We have seen how CCG accounts for unbounded extraction in surface syntax, using the rules of composition and type-raising to link extracted arguments to their underlying, canonical positions in the string. Of course, certain constituents do not permit such extraction and form "islands" from which arguments cannot escape (cf. Ross, 1967). For example, it is widely held that extraction is not possible from adjuncts, such as the adverbial *without reading the articles*. Thus the following relative clause is ungrammatical:

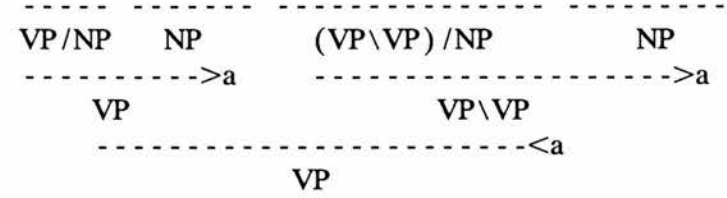
(18) * the articles which I will file letters without reading

CCG offers a natural characterisation of the island status of *without reading the articles*, since, like other adjuncts, it bears a leftward-modifying category:

---

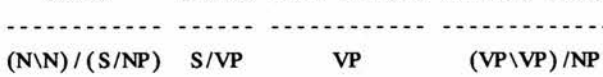
[4] In fact, the situation is a little more complicated than this, for although function composition is an associative operation, function application is not. This has not mattered in the examples considered so far, but in general the presence of certain higher-order categories, together with the non-associativity of function application, can induce semantically distinct readings by different orders of function application. We will return to this issue, and its implications for processing, in Chapter 6.

(19) file letters without reading the articles



This automatically prevents the construction of a bridge from the extracted element to its canonical position in the adjunct:

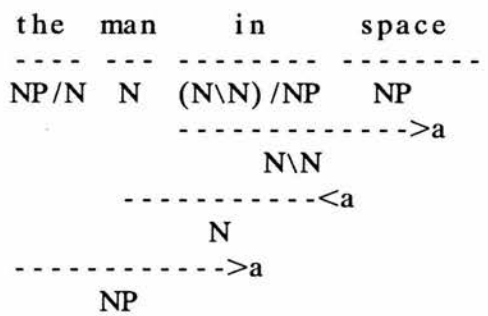
(20) * (the articles) which I will file letters without reading



Since *without reading the articles* is not an argument of the verb, *file letters* and *without reading* cannot combine. The Complex Noun Phrase Constraint of Ross (1967) can be captured in a similarly direct fashion, as relative clauses and PPs bear the noun-modifying category N\N.

While explaining some constraints on movement, this approach to adjuncts threatens the usefulness of CCG for incremental processing. In (19) a processor has to analyse the adverbial *without reading the articles* in isolation from the preceding material, connecting it to the VP only once it is completely analysed. Similarly, the complex NP *the man in space* is given the following non-incremental, right-branching analysis:

(21) the man in space

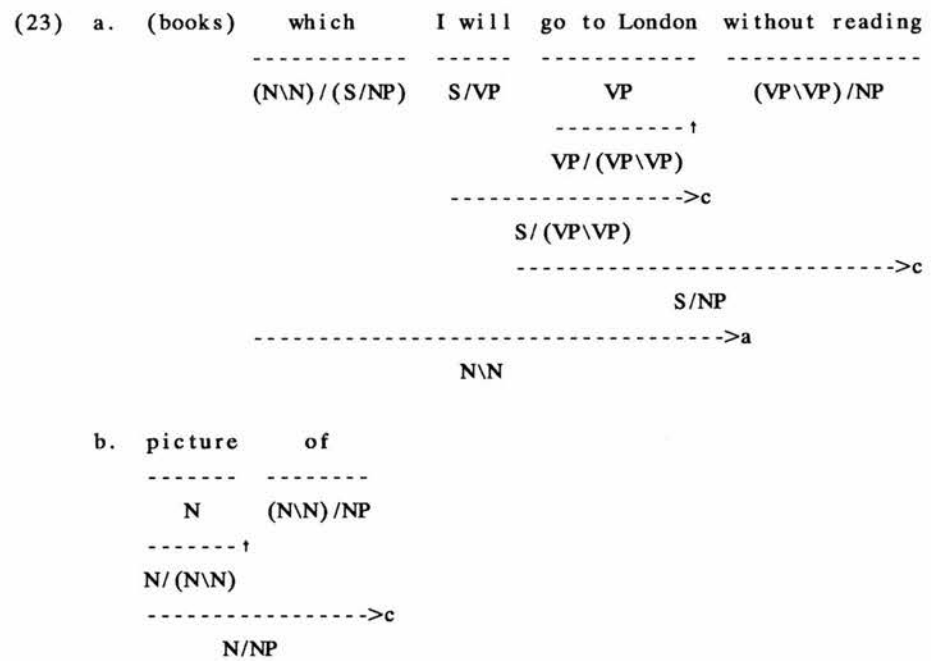


Incremental processing requires function composition, but composition allows extraction, and extraction appears to be forbidden in such cases.

However, it is interesting to note that extraction is not always forbidden from these syntactic classes. The following are well-known counter-examples to a constraint which prevents extraction from adverbials (22a) and complex NPs (22b):

(22) a. (I know there will be) some books which I will go to London without reading.
b. Who did you see a picture of?

And as Steedman (1987b, p421) points out, CCG can accommodate an alternative standpoint on island constraints, adopting these exceptions as the rule. The theory's general type-raising schemata (Steedman, 1987b, p413) in principle permit verb phrases and nouns to be type-raised over their modifiers, so sanctioning the extractions as follows:



Type-raising makes the adverbial in (23a) an argument of the VP, and allows the rule of function composition to connect the extracted item with its site in the canonical order. The same goes for (22b), with nominal type-raising and function composition importantly allowing the combination in (23b).

The theory of CCGs therefore lets us choose whether or not to capture island constraints within the syntax. Steedman (1987b, p421) suggests that if nominal and verbal type-raising are incorporated into the grammar in the above fashion, we could possibly turn to semantics for the constraints which must block clearly unacceptable examples like (18). He notes that the island constraints appear to be sensitive to semantic factors, and might be explicable in terms of the relative "reasonableness" of the semantic concepts involved.

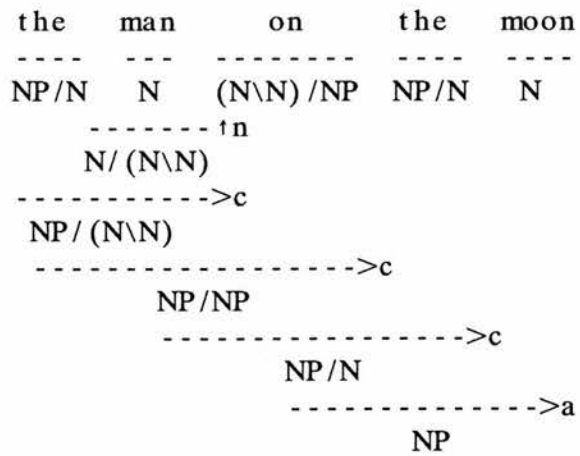
The general answer to this question is unresolved, but in the interests of incremental processing our particular CCG will include the additional rule of nominal type-raising in (24).

(24) Nominal Type-Raising ("↑n")

$N : x \quad \uparrow \quad N/(N\backslash N) : \lambda f.(f\ x)$

This will allow complex NPs to be incrementally analysed as in (25):

(25)

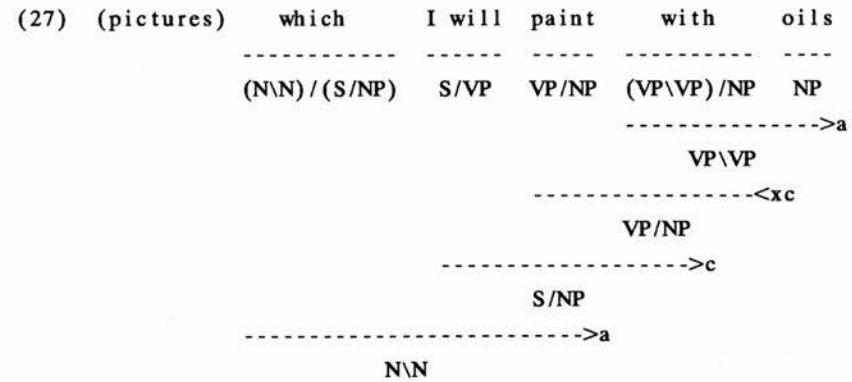


The syntactic analysis of (25) is unconventional in two respects. First, as discussed above, nouns type-raise over their modifiers. Second, we allow composition into NPs, and so give status to fragments such as *the man on the*. While providing an account of sentences like (22b), it should be emphasised that such freedom in the grammar also permits overgenerations like (26),

(26) * the trees which I painted the hill with

in which the object *the trees* of a complex NP *the hill with the trees* has been extracted from its canonical site. However, for the purposes of this thesis, we will put these wider syntactic ramifications to one side in order to make progress on the central issue of incremental interpretation.

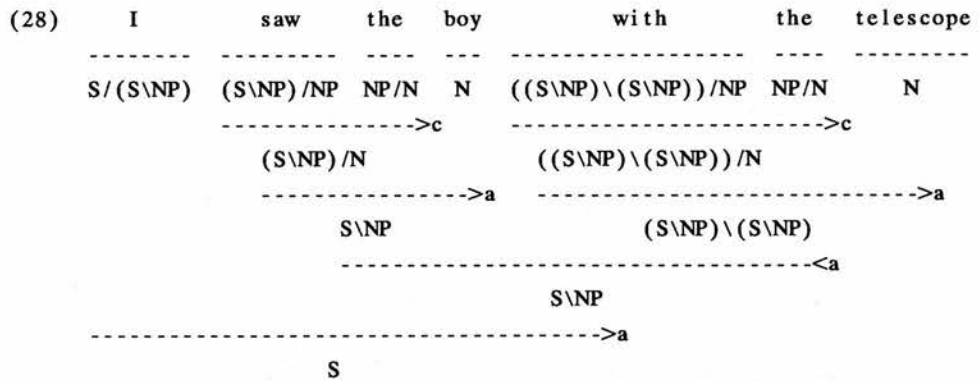
However, we will not bother to include a rule of verbal type-raising, as used in (23a) above. Whereas composition into noun-modifiers will prove to be crucial in accounting for the reference of the problematic NP *the rabbit in the hat* discussed in Chapter 1, less hinges on the ability to analyse adverbials incrementally. Furthermore, although verbal type-raising will allow composition into adverbials in examples like (19), it will not in other cases, such as (27):



In (27) the object of the VP has been extracted, and the rule of crossed backward composition (7) is invoked to form the relevant bridge. Here not even a lexical version of verbal type-raising applied to *paint* would do anything to change the fundamentally right-branching nature of this analysis.

This discussion does, of course, raise the more general question of whether CCG is really suitable for incremental interpretation, in the narrow rule-to-rule sense discussed earlier. It has clearly brought us a good way towards the goal, and the rules of composition and type-raising give a categorial grammar much more room for maneouvre than one without. But should one wish to do so, there are two obvious ways to produce a greater degree of left-to-right incrementation of the semantics. The first is to relax the grammar even more than we have done here, incorporating more type-raising rules and possibly even more combinatory rules. This move would incur a natural penalty in terms of the grammar's ability to constrain word order. The alternative is to relax the one-to-one correspondence between syntax and semantics, allowing the semantics to proceed more incrementally than the grammar rules themselves permit. Indeed, such a move has been advocated by Fenstad *et al* (1985) and Halvorsen (1986, 1987), albeit in a different framework and to somewhat different ends than our own. The penalty here is the perspicuity which may be lost in untying syntax and semantics in this way.

In those examples where our grammar does not permit a completely incremental analysis, we will be interested in the *optimally incremental* reading allowed. So for a sentence such as *I saw the boy with the telescope*, in the case where the PP *with the telescope* modifies the verb we will be most interested in the following derivation:



We conclude this section with a note on the location of the subject and nominal type-raising rules used in the system. Are they to be applied to *both* lexical categories *and* those categories output by the combinatory rules, or is type-raising a lexical rule alone? This remains an open question (cf. Steedman, 1987b, p421), but for the purposes of incremental interpretation and parsing it will simplify matters if we treat type-raising as a purely lexical operation. (For example, by performing type-raising as a preprocess in the lexicon, we do not have to consider type-raising during the parsing process.) Thus, for the case of subject type-raising, all NPs in the lexicon bear the additional category S/(S\NP). Furthermore, all lexical functions into NP are additionally stated as functions into S/(S\NP). So determiners bear two categories: NP/N and (S/(S\NP))/N.[5]

3.8. Parsing

For a substantial part of this thesis we will assume a very simple processing model for CCG. This processor works through the string from left to right, building syntactic and semantic descriptions by a bottom-up invocation of the combinatory rules. The processor combines constituents as early as possible, and so favours incremental analyses of strings.

---

[5] Unfortunately, such lexical type-raising creates a minor difficulty with the general application of nominal type-raising. Applying the rule of (24) to a noun such as *man* yields the following lexical entry:

(29) man := N/(N\N)

This definition means that the noun subcategorises for one noun-modifier. Now, the rule of (24) should also be invoked for any functions *into* N; in particular, it should be applied to the N embedded in (29), for *man* may expect more than one modifier. This, then, will first yield the category (N/(N\N))/(N\N) — the case where *man* expects two noun-modifiers; and then recursively produce a category subcategorising for three modifiers, and so on. The problem is that an arbitrary number of noun-modifiers may attach to any given head noun, and yet a limit must be placed on the number of raised categories in the parser's lexicon. The fragment of grammar implemented in Appendix A.2 therefore limits each noun to three categories, subcategorising for zero, one and two noun-modifiers.

Following Ades and Steedman (1982), we can achieve this effect with a non-deterministic variant of a shift-reduce parser consisting of just a push-down stack and a buffer (cf. Aho and Ullman, 1972). The input string is stored in the buffer, and syntactic and semantic information associated with words in the buffer is shifted, periodically, to form an entry on the stack. From time to time the stack is reduced, by applying a combinatory rule to the topmost stack items.

The procedures in Figure 3.1 give a more precise picture of the algorithm, which, for ease of exposition, is specified as a recogniser rather than a parser. So, as far as this algorithm is concerned, an entry on the stack consists of a syntactic category alone (though we will, in fact, add semantics in Chapter 5). Note also that the algorithm is presented as an NP recogniser, since the thesis only considers the semantic evaluation of noun phrases.

As indicated in Figure 3.1, the parser starts out with an empty stack and an input buffer containing the string of words to be processed. The main loop of the algorithm, the

---

Initialise. Set the stack to nil; fill the buffer with the input string. Call Parse.

Parse. If the input buffer is empty, and the stack contains a single entry, and this entry is the category NP, then report success; else do either step (1) or step (2):

(1) If there are at least two entries on the stack, then Reduce.

(2) If there is at least one lexical item in the input buffer, then Shift.

Reduce. Do steps (1)-(5):

(1) Pop the top entry C2 from the stack.

(2) Pop the next entry C1 from the stack.

(3) Find a rule s.t. C1 C2 ==> C.

(4) Push the entry C onto the stack.

(5) Call Parse.

Shift. Do steps (1)-(4):

(1) Remove the next lexical item from the input buffer.

(2) Find a lexical entry for the item with category C.

(3) Push the entry C onto the stack.

(4) Call Parse.

Figure 3.1. A non-deterministic shift-reduce parsing algorithm for CCG.

---

routine Parse, considers the state of the stack and buffer at each stage and either reports success or carries on parsing. In the latter case, depending on the current configuration it can either reduce the stack or shift another item from the input buffer. Both of these operations — Reduce and Shift — call Parse recursively once they have performed their respective operations. (We assume that a reduction fails if no rule matches the stack in step (3) of Reduce, and similarly that a shift action fails if no lexical entries can be found for the word in step (2) of Shift.)

Three sorts of non-determinism arise in the model of Figure 3.1. First, at each stage in the process, the parser potentially has a choice between reducing the stack and shifting from the buffer; let us call this a *reduce-shift* choice. (In Figure 3.1, this is represented in the procedure Parse by the choice between step (1) and (2).) Second, where a word is given a number of definitions in the lexicon (i.e. where it is *form-class ambiguous*), there will be a choice of categories to shift onto the stack; this is a *shift-shift* choice. (In Figure 3.1, this is represented in the procedure Shift by the instruction at step (2) to "find a lexical entry", given that there may be a number of such entries.) Third, more than one combinatory rule may match the two particular categories at the top of the stack; so this is a *reduce-reduce* choice. (In Figure 3.1, this is represented in the procedure Reduce by the instruction at step (3) to "find a rule".)

Chapter 6 addresses the problems of intelligently exploring these alternatives, and until then we will not worry about grammatical ambiguity. For the meantime we assume that the processor just happens to shift the correct lexical definition for a word in the string, and that it just happens to resolve reduce-shift conflicts in a way which gives the string an optimally incremental reading (cf. the discussion of optimal incrementality in the previous section). Thus, for the time being, we invest our parser with a magical property of foresight, which, in the case of reduce-shift non-determinism, allows it to make the maximum number of early reductions permitted by the global syntax of the string. By accident, reduce-reduce options seem not to arise from the combinatory rules employed here (see Table 3.1 below), and so we will not concern ourselves with these until Chapter 6.

3.9. Summary

In summary, combinatory grammar is comprised of two main units: a categorial lexicon and a set of combinatory rules. In the above sections, we reviewed just those rules which will be relevant in this thesis — function application, and certain instances of

Combinatory Rules

Forward Application(">a")

X/Y : f Y : y ==> X : (f y)

Backward Application("<a")

Y : y X\Y : f ==> X : (f y)

Forward Composition(">c")

X/Y : f Y/Z : g ==> X/Z : λz. [f (g z)]

Generalised Forward Composition (">c")

X/Y : f (Y/W)/Z : g ==> (X/W)/Z : λz.λw. [f (g z w)]

Crossed Backward Composition("<xc")

Y/Z : g X\Y : f ==> X/Z : λz. [f (g z)]

Type-Raising Rules

Subject Type-Raising("ts")

NP : x ↑ S/(S\NP) : λf.(f x)

Nominal Type-Raising("tn")

N : x ↑ N/(N\N) : λf.(f x)

Table 3.1. Summary of combinatory and type-raising rules.

function composition. The grammar is supplemented by several type-raising rules for converting arguments into functions, and we chose to invoke these as lexical rules. Table 3.1 lists the complete set of combinatory and type-raising rules which will be used in the thesis. In addition, we have seen how the rules of function composition and type-raising may be used to produce incremental analyses of strings, and yet yield semantic translations that are logically equivalent to those that would be derived by function application alone. Finally, a simple non-deterministic version of a shift-reduce parser has been specified, for parsing input strings with respect to the grammar.

Chapter 4

Reference Evaluation as Network Consistency

We now turn to the theory underlying the procedures for reference evaluation in this thesis. Recall from Chapter 1 that our interest is in singular noun phrases which refer to specific entities already known to the hearer of a discourse. The particular algorithm we adopt for this purpose is one of a general class developed by Mackworth and others for enforcing consistency in a network of relations. We came across the essential spirit of these algorithms when discussing Mellish's constraint-oriented process of reference evaluation in Chapter 2; they work by "filtering", or refining, candidate sets of values in accordance with a collection of known constraints.

However, before considering algorithms for network consistency, we take a step back and define what is meant by reference to the context. The next section shows how reference may be viewed as a *constraint satisfaction problem*, regardless of the method for resolving the reference. Accordingly, the subsequent section sketches some generally accepted mechanisms for solving such problems in order to set the scene for our chosen method, Mackworth's network consistency algorithms. Thus, in these terms, Winograd and Mellish both implicitly regard reference to a known entity as a constraint satisfaction problem, but they employ different mechanisms for approaching the problem: Winograd uses a backtracking search process provided by the Planner language, whereas Mellish uses a filtering algorithm similar to those of Mackworth. (The reader should note that this distinction is frequently blurred in the AI literature, and that the term "constraint satisfaction" is often used more specifically to refer to an approach which, here, we are calling network consistency.)

We then consider the network consistency approach to constraint satisfaction problems, and the details of the particular consistency algorithm used for reference evaluation in this thesis. We also discuss the relevance of this process to *incremental* reference evaluation, the subject of Chapter 5.

The chapter concludes with an investigation into the adequacy of this technique for noun phrase evaluation. This investigation is necessary because network consistency algorithms are not as powerful as conventional search methods, such as backtracking, and

provide only a limited picture of the space of possible solutions to a given constraint satisfaction problem. This final section therefore states two specific requirements of our reference evaluator and, by virtue of a result due to Freuder, shows that the network consistency algorithm presented earlier will always meet these requirements for the class of referring expression addressed by the thesis. This section also points to some cases, beyond the scope of the thesis, where the algorithm will not be enough.

Much of the following material is taken from papers by Mackworth, Freuder and others (although they do not discuss the relevance of their techniques to linguistics); in particular, Mackworth (1977a, 1977b, 1987), Mackworth and Freuder (1985), and Freuder (1978, 1982). The reader is referred to these articles if in need of further background or detail.

4.1. Characterising Reference as a Constraint Satisfaction Problem

We are considering reference to known entities in a context, and so it will be helpful to start by reviewing what we mean by a context. Recall from Chapter 1 that the term *context* is being used to describe a hearer's mental model of the domain of reference, this consisting of those entities and relations which have been made salient in the hearer's mind, through either linguistic or non-linguistic means. For our purposes, this amounts to a finite set of facts, each fact consisting of a first-order predicate applied to one or more individuals or *entities*. For example, the following context involves certain people visiting particular places:

(1) { man(man1), man(man2), man(man3),
town(town1), town(town2), town(town3),
visit(man1,town1), visit(man1,town3), visit(man2,town2) }

So this context involves the predicates *man*, *town* and *visit* applied to various men and towns. In total, the context introduces the set of entities {man1,man2,man3,town1,town2,town3}. (Later on we will index some formulae with the event they denote, and these event indices will also be seen as entities in the context.) We will make an assumption akin to the closed-world assumption, and so as far the hearer is concerned, *man3* does not go anywhere.

Now let us consider the following definition of a constraint satisfaction problem, from Mackworth (1987, p206; 1977a, p99). A boolean *constraint satisfaction problem* (CSP) consists of a set of *variables*, each of which must be instantiated in a particular *domain* of values, and a set of *predicates* which the values of the variables must simultaneously satisfy.

We can schematise such a problem as the formula

(2) $(\exists e_1)\ (\exists e_2)\ \ldots\ (\exists e_n)\ (e_1 \in D_1)\ (e_2 \in D_2)\ \ldots\ (e_n \in D_n)\ P(e_1,e_2,\ldots,e_n)$

where $P(e_1,e_2,\ldots,e_n)$ abbreviates an arbitrary conjunction of predicates over subsets of the variables $\{e_1,e_2,\ldots,e_n\}$; and where each of the variables $\{e_1,e_2,\ldots,e_n\}$ is associated with a corresponding value domain from $\{D_1,D_2,\ldots,D_n\}$. Given an instance of a general constraint problem in (2), the task may be to determine its truth-value, or to find which n-tuples of values satisfy it.

Many tasks in AI can be seen as constraint satisfaction problems, and our particular interest is in resolving the reference of singular noun phrases which are anaphoric to a certain context. How does the general problem in (2) relate to the problem of reference? The relationship is straightforward: the value domains will be made up of entities in the context, and the predicates and variables derived from the senses of the words in a phrase. Each variable in the CSP can be thought of as a reference which must be resolved to one or more potential referents in its domain of possibilities. The resolution must be in line with the predicates which, together with the context, constrain the possible assignments of values.

Consider, for instance, the problem of determining the entities involved in the reference of the indefinite NP in (3).

(3) a man who visits a town

We can represent this as a CSP in the style of (2):

(4) $(\exists e_1,e_2)\ (e_1 \in D_1)\ (e_2 \in D_2)\ man(e_1)\ \&\ visit(e_1,e_2)\ \&\ town(e_2)$

D_1 and D_2 will be the same set, the set of entities in the context. So, in the context of (1), $D_1 = D_2 = \{man1,man2,man3,town1,town2,town3\}$. Satisfaction of the formula will involve assigning e_1 and e_2 values from this set, and seeing whether the instantiated predicates coincide with the formulae in the context.[1]

In general, we will assume that all variables in reference problems start out with the same value domain, namely the set of all entities in the context. (A more sophisticated approach to the semantics might *sort* the variables into classes such as *person* or *place*, and

[1] Definite NPs are seen as determining CSPs in exactly the same fashion, and so the expression in (5)

(5) the man who visits the town

also derives the CSP in (4). However, in order to avoid the complicating issue of the unique reference presupposed by a definite expression, which will be addressed in Chapter 5, this chapter will continue to use indefinite expressions for examples.

so automatically assign a variable a particular domain of values according to its sort. Within the limited scope of this project, though, variable sorting would bring no obvious benefits.)

From now on we will use the term *constraint* for any atomic predicate in the body of (2). Thus each of the atomic conjuncts in (4) — *man*(e_1), *visit*(e_1,e_2), and *town*(e_2) — is a constraint. In addition, we will say that a given constraint *constrains* or is a *constraint on* some given variable iff the variable is one of its arguments. So the constraint *visit*(e_1,e_2) constrains, and constrains only, the variables e_1 and e_2. (Sometimes, for emphasis, we will say "directly constrains" instead of "constrains".)

We will occasionally use terminology more oriented towards linguistics in describing a CSP and its parts. For example, the term *logical form* will be used to refer to a CSP like (4). And the domains associated with the logical form of a constituent will be seen as its *extensions*. Similarly, the variables occurring in a CSP will sometimes be called *semantic variables* or *extension variables*.

4.2. General Methods for Solving Constraint Satisfaction Problems

What procedures are available to solve CSPs? Although we will adopt Mellish's proposal to use network consistency techniques, it is worth surveying some other potential solution schemes in order to set the stage.

A basic way to solve the CSP in (2) is to *generate-and-test* potential solutions. This involves computing the Cartesian product D of all the given value domains, i.e.

$$D = D_1 \times D_2 \times \ldots \times D_n,$$

and then evaluating the body of the formula in (2) on each n-tuple in D. One may choose to stop at the first solution, or produce all satisfying n-tuples.

Backtracking is a more sophisticated search strategy, and will be familiar to many as the control structure underlying the programming language Prolog. A backtracking algorithm explores the product space D by sequentially instantiating variables in the body of (2). Once a constraint is fully-instantiated, its truth-value is determined; if it is false, the last variable with untried values in its domain is instantiated with another value. Note that Winograd's SHRDLU program uses this method for finding the referents of noun phrases. There a referring expression translates into a conjunction of goals in the language Planner; and the interpreter attempts to satisfy such goals by a depth-first exploration of the space of possible assignments of values to variables, backtracking on failure.

Network consistency algorithms offer a different approach altogether. A *consistency algorithm*, in Mackworth's use of the term, regards the problem as one of elimination rather than search. The goal is to refine the value domains as much as possible so that values which cannot be part of a global solution to the problem are eliminated. So rather than *searching* for a solution, a consistency algorithm *eliminates* what cannot be a solution. It does this by ensuring that the values in each variable's domain are consistent with the constraints on that variable. There are varying strengths of consistency that can be imposed on a CSP, depending on how many domains and predicates are considered simultaneously. Of course, Mellish's reference process has already given us an example of such a consistency algorithm in action.

This latter perspective is the one we will adopt. It offers an inherently incremental process, natural for the evaluation strategy we are interested in, and a later section will suggest that in many cases full search procedures may be unnecessary for reference evaluation. The following sections therefore set out to define our terms and present the particular consistency algorithm employed in this thesis.[2]

4.3. Network Consistency: Basic Concepts

The best way to understand the workings of a consistency algorithm is to view the CSP as an undirected graph in which the vertices stand for variables and each edge between vertices i and j represents a constraint on the variables e_i and e_j. The graph is annotated as follows. Edges are labelled with the predicates they represent, and the vertices are associated with the corresponding variable's domain of values (forming *nodes*). Unary predicates are drawn as loops on the vertices. We will refer to such an annotated graph as a *constraint network*.

For example, consider the indefinite noun phrase *a man who visits a town on a river*, where the PP *on a river* is assumed to modify *town*. The CSP for this expression, in (6), has the constraint network in Figure 4.1.

(6) $(\exists e_1,e_2,e_3)\ (e_1 \in D_1)\ (e_2 \in D_2)\ (e_3 \in D_3)$
$\text{man}(e_1)\ \&\ \text{visit}(e_1,e_2)\ \&\ \text{town}(e_2)\ \&\ \text{on}(e_2,e_3)\ \&\ \text{river}(e_3)$

The constraint network models the three variables in the constraint problem with three nodes, each consisting of a vertex i and the associated variable e_i and domain D_i. The

[2] Formal theorem-proving methods are another candidate solution scheme, but will not be explored here; see Hobbs (1986) and Charniak and Goldman (1988) for some recent applications of these techniques to discourse-level problems.

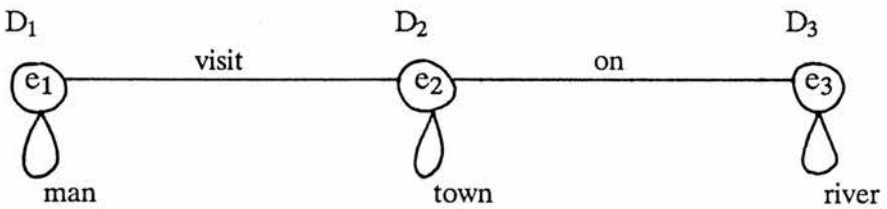


Figure 4.1. An undirected constraint network.

binary predicates {visit,on} figure as edges between the nodes they constrain, and the unary predicates {man,town,river} are indicated by loops on the appropriate nodes.[3]

From the point of view of defining consistency, it will be convenient to make the edges in such a network *directed*. This is easily done. For every edge between nodes i and j in the undirected graph, we draw two directed arcs, one from i to j and the other from j to i. So Figure 4.1 appears as the directed graph of Figure 4.2.

Although we will not always draw constraint problems as graphs, the network model serves to motivate some of the ideas behind consistency and the related notion of constraint propagation. Following Waltz (1975) and Montanari (1974), Mackworth (1977a) argues that three degrees of consistency are required in a constraint network, corresponding to its

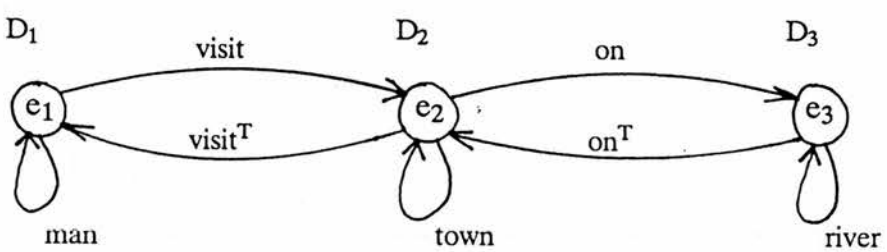


Figure 4.2. A directed constraint network

[3] The graphical representation of unary predicates used here is perhaps somewhat misleading. For example, in Figure 4.1, representing the unary predicate *man* as a loop, i.e. as an edge from node 1 to node 1, might be taken as suggesting that there is in fact a binary constraint $man(e_1,e_1)$ on this node. However, this representation is standard practice and so we will continue to use it here.

nodes, arcs, and paths. Node, arc, and path consistency are differing states of consistency in which progressively more information from the constraints is reflected in the value domains. Only the first two sorts of consistency will be used for our reference-based constraint problems, and so we will pay most attention to node and arc consistency.

First, then, node consistency. A node is *node consistent* iff every value in its domain satisfies its constraining unary predicate. So, a node i is node consistent iff

$$(\forall a)\ [a \in D_i] \supset P_i(a)$$

where P_i is the unary predicate constraining node i. Thus, in Figure 4.2, node 1 is node consistent iff D_1 consists only of men. More formally, node 1 is node consistent iff

$$(\forall a)\ [a \in D_1] \supset man(a)$$

Arc consistency similarly defines consistency for the directed arcs in a network. An arc from i to j is *arc consistent* iff for every value in the domain D_i there is at least one value in D_j such that the pair of values satisfy the constraining binary predicate. Thus arc (i,j) is arc consistent iff

$$(\forall a)\ [a \in D_i] \supset (\exists b)\ [(b \in D_j)\ \&\ P_{ij}(a,b)]$$

where P_{ij} is the binary predicate labelling the arc from i to j. So, in Figure 4.2, arc (1,2) is arc consistent iff each of the entities in D_1 visits at least one of the entities in D_2. More formally, arc (1,2) is arc consistent iff

$$(\forall a)\ [a \in D_1] \supset (\exists b)\ [(b \in D_2)\ \&\ visit(a,b)]$$

In contrast, arc (2,1) is arc consistent iff each element of D_2 is visited by at least one entity in D_1, i.e. iff

$$(\forall a)\ [a \in D_2] \supset (\exists b)\ [(b \in D_1)\ \&\ visit^T(a,b)]$$

The constraint $visit^T(a,b)$ is just visit(b,a), since for any predicate we assume that $P^T(x,y) \equiv P(y,x)$.

It is worth mentioning path consistency here, for we will return to it in the final section. A path of length 2 from node i through node m to node j is path consistent iff

$$(\forall a)\ (\forall c)\ [(a \in D_i)\ \&\ (c \in D_j)\ \&\ P_{ij}(a,c)] \supset (\exists b)\ [(b \in D_m)\ \&\ P_{im}(a,b)\ \&\ P_{mj}(b,c)]$$

where P_{ij} labels the arc from i to j, P_{im} labels the arc from i to m, and P_{mj} labels the arc from m to j. Thus, path consistency simultaneously considers the consistency of two adjacent nodes i and j with respect to a third node m which lies on a path between i and j. The path is path consistent if for all pairs of values $<a,c> \in D_i \times D_j$ which satisfy the binary predicate directly constraining i and j, P_{ij}, there exists a third value $b \in D_m$ such that the

values together satisfy the predicates P_{im} and P_{mj}.

Since we employ only node and arc consistency, we will say that a network is *consistent* iff all its nodes and arcs are consistent. However, clearly the initial state of a network like that in Figure 4.2 is typically both node and arc *inconsistent*. The job of a consistency algorithm is to make it consistent. Enforcing node consistency is easy: just remove those values which do not satisfy the constraining unary predicate. In the case of node 1, then, the following operation of domain refinement will ensure node consistency:

$$D_1 \leftarrow \{a \mid (a \in D_1) \ \& \ man(a)\}$$

Similarly, the way to make an arc (i,j) arc consistent is to remove elements from the domain of its *initial* node i until the arc is consistent.[4] This amounts to checking each value in D_i to see if it has a corresponding value in D_j so that, together, they satisfy the constraining predicate; if there is no corresponding value in D_j, then the value in D_i can be deleted. So arc (2,1) in Figure 4.2 could be made arc consistent by resetting D_2, as in

$$D_2 \leftarrow \{a \mid (a \in D_2) \ \& \ (\exists b) \ [(b \in D_1) \ \& \ visit(b,a)]\}$$

meaning that D_2 consists only of the entities {a} visited by entities {b} in D_1.

Making a network consistent therefore involves making all of its nodes and arcs consistent. Node consistency can be enforced by a single sweep of the network, applying the operation of domain refinement to each node with respect to its unary predicate. Node consistency does not have to be checked again; once a node is node consistent, it will always be node consistent. The same is not true for the arcs. Even if arc (i,j) is currently arc consistent, it may later become inconsistent again if some arc (j,k) is subsequently revised. This is where *constraint propagation* will be useful, as a means of knowing which nodes might be affected by a particular domain refinement.

So the consistency of a complete network depends entirely on the consistency of its nodes and arcs. Consistency is therefore a *local* notion; it is defined locally for the parts of a network, and the consistency of a complete network is just a trivial sum of the consistency of its parts. Unlike search procedures like backtracking, a consistency algorithm does not engage in a global analysis of the problem; it simply iterates consistency operations on the arcs in a network until it can go no further.

---

[4] The terminology here is adapted from graph theory. Given an arc (i,j), we say node i is its *initial node* and node j is its *terminal node*. Furthermore, if node i and node j are related by arc (i,j), i is said to be *adjacent to* j, and j *adjacent from* i.

4.4. A Network Consistency Algorithm using Constraint Propagation

We will now formalise the above notions of consistency, domain refinement, and constraint propagation so that an algorithm can be specified to determine a consistent (or inconsistent) network for a constraint problem such as (6). This formalisation will in fact subsume node consistency under a generalised version of arc consistency which applies to n-ary predicates. Given a CSP such as (6), for the purposes of this section we will assume an *arc* is simply a pairing of a variable and *any* constraint on that variable; regardless of whether the constraint is unary, binary, or, indeed, n-ary. Thus, arcs are derived from the body of a CSP according to the definition in (7):

(7) If e_i is a variable and C is a constraint on e_i, the pair $<i,C>$ is an *arc*

So a network is created for a CSP by pairing each variable in the problem with each constraint on that variable. The problem in (6), for example, has a network represented by the following arc set:

(8) $\{<1,man(e_1)>, <2,town(e_2)>, <3,river(e_3)>,$
$<1,visit(e_1,e_2)>, <2,visit(e_1,e_2)>,$
$<2,on(e_2,e_3)>, <3,on(e_2,e_3)>\}$

We now say an arc $<i,C>$ is arc consistent when the domain of its initial node i is consistent with each of the domains of the other nodes implicit in C. Thus the following definition of arc consistency subsumes node consistency, since node consistency is just the special case of arc consistency where i is the only node constrained by C.

(9) Arc Consistency (n-ary predicates)

Arc $<i,P(e_1,\ldots,e_i,\ldots,e_n)>$ is arc consistent iff

$$(\forall a_i)\ [a_i \in D_i] \supset (\exists a_j \in D_j, j = 1,\ldots,n, i \neq j)\ P(a_1,\ldots,a_i,\ldots,a_n)$$

That is, a node i is *arc consistent* with some constraint $P(e_1,\ldots,e_i,\ldots,e_n)$ iff for every value a_i in its domain D_i there is at least one value a_j in the domain D_j of each of the other constrained variables e_j such that the values simultaneously satisfy the predicate P. If P is unary, e_i is the only variable requiring instantiation.

Domain refinement is defined in a similarly general fashion. An arc $<i,P(e_1,\ldots,e_i,\ldots,e_n)>$ is made consistent by removing from D_i all values which have no corresponding values in D_j ($j = 1,\ldots,n$, $i \neq j$) such that P is satisfied. In other words, the way to make an arc consistent is to remove elements from the domain of its initial node i until the condition in (9) holds. We can therefore enforce node and arc consistency by the single operation of domain refinement in (10):

(10) Domain Refinement (n-ary predicates)

REFINE(i, $P(e_1,\ldots,e_i,\ldots,e_n)$)
$= \{a_i \mid (a_i \in D_i)\ \&\ (\exists\, a_j \in D_j, j = 1,\ldots,n, i \neq j)\ P(a_1,\ldots,a_i,\ldots,a_n)\}$

(Adapted from Davis, 1987, p285)

For example, returning to the network in (8) (represented graphically in Figure 4.2), node 1 is made node consistent by the operation

$\text{REFINE}(1, \text{man}(e_1)) = \{a \mid (a \in D_1)\ \&\ \text{man}(a)\}$

To take another example, the arc from node 2 to node 1 in Figure 4.2 could be made arc consistent by refining D_2 so that

$\text{REFINE}(2, \text{visit}(e_1,e_2)) = \{a \mid (a \in D_2)\ \&\ (\exists\, b)\ [(b \in D_1)\ \&\ \text{visit}(b,a)]\}$

Given this fundamental operation of domain refinement, we may formalise the notion of *constraint propagation*, which, following Waltz (1975) and Mackworth (1977a), can be used to ensure the consistency of a complete network. Suppose we are adding a new constraint to a consistent network. Informally, a constraint propagation algorithm starts at the nodes in this constraint and makes them consistent. If this involves eliminating some domain values, then any neighbouring nodes which might be affected by the modification are re-checked for consistency. If their domains change, then the effects will spread wider still. The implications of one alteration in state gradually propagate around the network until it is consistent again.

The first step in processing a general CSP is to derive its set of arcs. Each arc must then be checked for consistency. In addition, whenever a domain is refined, the consistency of those arcs which might be affected by the change must be checked *again*. The innovation of Waltz's algorithm is that if elements are removed from a node's domain, only the immediately adjacent nodes need to be re-considered for consistency. Mackworth (1977a,1977b) generalises Waltz's method as follows.

Suppose that node i is refined with respect to constraint C so that the assignment

$D_i \leftarrow \text{REFINE}(i,C)$

reduces the size of D_i, but D_i remains non-empty. Which other arcs might have been made inconsistent as a result of this change? The following three restrictions collectively determine the set of arcs $\{<j,C'>\}$ that must be checked again:

(i) Only those arcs $\{<j,C'>\}$ which constrain node i need to be re-considered — any others cannot be directly affected by the change because their value domains do not directly depend on D_i.

(ii) Any arcs *from* node i, $\{<i,C'>\}$, can be disregarded. Refining D_i with respect to C cannot *directly cause* the inconsistency of other arcs $<i,C'>$. If $<i,C'>$ was previously consistent, then removing elements from D_i can only preserve that consistency. Thus only arcs which lead *to* i need to be reviewed.

(iii) Any arcs which lead to i with the same constraint C, $\{<j,C>\}$, can be disregarded. No $<j,C>$ could become inconsistent as a *direct result* of the revision of $<i,C>$, for their consistencies depend on the same constraint holding. To paraphrase Mackworth (1977a, p105), "the deletions in D_i were made precisely because there were no corresponding values in D_j."

The algorithm in Figure 4.3 incorporates this notion of constraint propagation by maintaining a queue of arcs awaiting the consistency check. The procedure is an adaptation of Mackworth's NC algorithm for consistency in a network of n-ary constraints (Mackworth 1977b, p603).[5] The first part of Mackworth's consistency algorithm produces a set of arcs, Arcs, from the constraints in the CSP (steps 1-2). These arcs — the complete network — then form the initial queue Q of arcs to be made consistent (step 3). The main loop of the algorithm, in step (4), refines domains and propagates the effects, Waltz-style. If REFINE ever reduces a domain to the empty set, the algorithm fails, since the problem

---

Mackworth. Do steps (1)-(4):

(1) Initialise the set Arcs to $\varnothing$.

(2) For each constraint C, do step (2.1):

  (2.1) For each argument e_i of C, add the pair $<i,C>$ to Arcs.

(3) Initialise Q to the set Arcs.

(4) While $Q \neq \varnothing$, do steps (4.1)-(4.3):

  (4.1) Remove the next arc $<i,C>$ from Q.

  (4.2) Reset $D_i \leftarrow$ REFINE(i,C).

  (4.3) If D_i was reduced in size by step (4.2), then do step (4.3.1):

    (4.3.1) If $D_i = \varnothing$ then FAIL, else reset

$$Q \leftarrow Q \cup \{<j,C'> \mid <j,C'> \in \text{Arcs}, C' \text{ constrains } i, i \neq j, C' \neq C\}$$

Figure 4.3. Mackworth's network consistency algorithm for n-ary constraints.

---

[5] NC is a generalised version of AC-3 (Mackworth 1977a, p106), for unary and binary predicates, and AC-3 is itself a generalisation of Waltz' original filtering algorithm (Waltz, 1975).

must be inconsistent. The existential quantification in (2) is therefore implicit, embedded in the algorithm. If REFINE otherwise changes a domain, the effects are propagated by placing any arcs $\{<j,C'>\}$, according to the restrictions detailed above, on the Q. (The conditions that C′ constrains i, $i \neq j$, and C′ $\neq$ C correspond to the above restrictions (i), (ii) and (iii), respectively.)

Propagation continues indefinitely, but the procedure will always halt. Intuitively, this is because the propagation queue only grows if some value domain is reduced. Otherwise the queue shrinks. Thus at some stage, either a domain is reduced to the empty set, in which case the procedure reports failure, or the queue empties completely, in which case the procedure has determined a consistent network.

We will now run through an example to illustrate Mackworth's algorithm for network consistency, considering the constraint problem of (6) in a context designed to illustrate the full power of constraint propagation. The context is listed in (11) below.

(6) $(\exists e_1,e_2,e_3)\ (e_1 \in D_1)\ (e_2 \in D_2)\ (e_3 \in D_3)$
$man(e_1)\ \&\ visit(e_1,e_2)\ \&\ town(e_2)\ \&\ on(e_2,e_3)\ \&\ river(e_3)$

(11) {man(man1),
man(man2),
man(man3),
man(man4),
man(man5),
man(man6),
woman(woman1),
town(town1),
town(town2),
town(town3),
town(town4),
town(town5),
village(village1),
river(river1),
river(river2),
lake(lake1),
visit(man1,town1),
visit(man1,town3),
visit(man2,town1),
visit(man2,town2),
visit(man3,village1),
visit(man5,town3),
visit(man6,town4),
visit(woman1,town5),
on(town1,river1),
on(town2,river1),
on(town3,lake1),
on(town5,river2),
on(village1,river2) }

Let us assume that the initial Q, consisting of all arcs derivable from (6), is ordered as follows:

$Q = [<1,man(e_1)>, <1,visit(e_1,e_2)>, <2,visit(e_1,e_2)>,$
$<2,town(e_2)>, <2,on(e_2,e_3)>, <3,on(e_2,e_3)>, <3,river(e_3)>\ ]$

In addition, assume that whenever arcs are *added* to the queue they are added to its *front*. Figure 4.4 lists the contextual entities which form the three initial domains D_1, D_2, and D_3. The number to the right of each entity indicates the point at which it is eliminated from its domain, in relation to the other values.

D_1		D_2		D_3	
man1		man1	12	man1	27
man2		man2	13	man2	28
man3	24	man3	14	man3	29
man4	11	man4	15	man4	30
man5	43	man5	16	man5	31
man6	26	man6	17	man6	32
woman1	1	woman1	18	woman1	33
town1	2	town1		town1	34
town2	3	town2		town2	35
town3	4	town3	42	town3	36
town4	5	town4	25	town4	37
town5	6	town5	19	town5	38
village1	7	village1	23	village1	39
river1	8	river1	20	river1	
river2	9	river2	21	river2	40
lake1	10	lake1	22	lake1	41

Figure 4.4. Order of elimination for problem (6) in context (11).

Given the initial queue above, arcs are processed in the following order:

(a) $<1,man(e_1)>$

Eliminations 1-10;

$Q = [<1,visit(e_1,e_2)>,<2,visit(e_1,e_2)>,<2,town(e_2)>,$
$<2,on(e_2,e_3)>,<3,on(e_2,e_3)>,<3,river(e_3)>]$

(b) $<1,visit(e_1,e_2)>$

Elimination 11;

$Q = [<2,visit(e_1,e_2)>,<2,town(e_2)>,<2,on(e_2,e_3)>,<3,on(e_2,e_3)>,<3,river(e_3)>]$

(c) $<2,visit(e_1,e_2)>$

Eliminations 12-22;

$Q = [<2,town(e_2)>,<2,on(e_2,e_3)>,<3,on(e_2,e_3)>,<3,river(e_3)>]$

(d) $<2,town(e_2)>$

Elimination 23;

$Q = [<1,visit(e_1,e_2)>,<2,on(e_2,e_3)>,<3,on(e_2,e_3)>,<3,river(e_3)>]$

(e) <1,visit(e_1,e_2)>

Elimination 24;

Q = [<2,on(e_2,e_3)>,<3,on(e_2,e_3)>,<3,river(e_3)>]

(f) <2,on(e_2,e_3)>

Elimination 25;

Q = [<1,visit(e_1,e_2)>,<3,on(e_2,e_3)>,<3,river(e_3)>]

(g) <1,visit(e_1,e_2)>

Elimination 26;

Q = [<3,on(e_2,e_3)>,<3,river(e_3)>]

(h) <3,on(e_2,e_3)>

Eliminations 27-40;

Q = [<3,river(e_3)>]

(i) <3,river(e_3)>

Elimination 41;

Q = [<2,on(e_2,e_3)>]

(j) <2,on(e_2,e_3)>

Elimination 42;

Q = [<1,visit(e_1,e_2)>]

(k) <1,visit(e_1,e_2)>

Elimination 43;

Q = []

The application of arc consistency to (6) thus labels e_1 with {man1,man2}, e_2 with {town1,town2}, and e_3 with {river1}.

4.5. Network Consistency and Incremental Processing

As the above example illustrates, the consistency algorithm works at a local level, monotonically removing impossibilities step-by-step. Bobrow and Webber (1980b, p5), Mellish (1985) and Davis (1987, p283) point out that this makes such network consistency algorithms well-suited to incremental systems which need to use partially-evaluated results.[6] Note that in the example the whole constraint problem was available to the

[6] In the terms of Bobrow and Webber (1980b), constraint propagation is an Incremental

procedure before it started. This is by no means essential: the procedure can start enforcing consistency as soon as the first constraint is known. As each additional constraint arrives, its arc set is computed and added to the existing network. The effects of any addition are propagated at once, so that at each stage the consistency (or, perhaps, inconsistency) of the network is determined.

In the realm of incremental reference evaluation, constraints can be derived from a noun phrase *as it is parsed*, and passed on-line to the propagation algorithm which assimilates them into a growing network. Each node in the network represents an under-specified extension, gradually being made more specific by the operation of domain refinement. Note that the incremental evaluation component does not have to wait for a head noun before it can begin evaluating an extension; in the above example it is the visiting relation, in step (c), which first refines the domain for the "town" variable, e_2.

4.6. The Adequacy of Network Consistency for Reference Evaluation

Let us consider the general means and objectives of network consistency, in the light of the example at the end of section 4.4. The algorithm works by removing local inconsistencies from the network that can never be part of a global solution; it eliminates what cannot be the case rather than searching for what might be. In the example, *town4* is deleted from D_2 because it violates one of its local constraints, that it should be "on" something. But the algorithm does not tell us exactly what the solutions to the problem *are*. Whilst a search procedure would find the set of satisfying tuples

$$S = \{<man1,town1,river1>,<man2,town1,river1>,<man2,town2,river1>\},$$

all we can infer from network consistency is that the set of solutions is some subset of the cross-product of the final value domains. So in this case we only know

$$S \subseteq D_1 \times D_2 \times D_3 = \{<man1,town1,river1>,<man1,town2,river1>,\\ <man2,town1,river1>,<man2,town2,river1>\}.$$

The failure to infer everything (about the solution set) from the results of network consistency comes from what Davis (1987, p287) calls the "narrow bandwidth" of domains as an interface between the network and any user of the system.

The vague results of network consistency leave two questions unanswered. First, is there a solution at all? In general, unless every domain is reduced to one element, there is

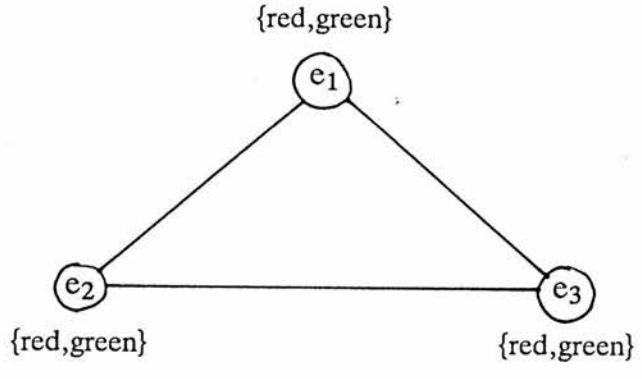
---

Description Refinement process which works with *disjunctive* rather than *abstract* under-specification. That is, here, the domain or "candidate set" of each variable represents a disjunction of potential referents.

no guarantee that a solution exists (Mackworth, 1987). Second, if the constraint problem *is* satisfiable, what exactly *are* the solutions?

In line with this discussion, we will say that a constraint satisfaction problem over variables $<e_1,e_2,\ldots,e_n>$ is *satisfiable* iff there exists some assignment of values $<a_1,a_2,\ldots,a_n>$ to the variables which makes the CSP true in the context. To see that network consistency is not necessarily co-extensive with satisfiability, consider an instance of the classical graph colouring problem from Freuder (1978). This amounts to colouring the vertices of a graph, using a given set of colours, such that no adjacent vertices have the same colour. As Figure 4.5 shows, we can directly represent such a problem as a constraint network: vertices in the graph are represented by nodes in the constraint network, labelled with domains consisting of the available colours, and each edge in the graph is represented by an edge (i,j) in the network indicating that i and j must have different colourings.

It is important to note that the remainder of this section will assume that the edges in a constraint network are *undirected* and thus follow the pattern of Figure 4.5, and Figure 4.1 above, rather than Figure 4.2 above. An undirected edge (i,j) is arc consistent if for every element in the domain of node i, D_i, there is an element in D_j such that the values satisfy the constraining binary predicate, and if for every element in D_j there is an element in D_i such that these values also satisfy the constraining binary predicate. We make this change because Freuder's results apply to undirected networks, rather than the directed



(Adapted from Freuder, 1978, p959)

Figure 4.5. An unsatisfiable graph colouring problem as a constraint network

ones we have assumed until now. (In addition, we will only consider unary and binary constraints from now on.)

Given that only two colours (red and green) are available for the complete three-vertex graph represented in Figure 4.5, the corresponding constraint problem is unsatisfiable: there is no instantiation of values $<a_1,a_2,a_3>$ to the variables $<e_1,e_2,e_3>$ which simultaneously satisfies all the constraints on those variables. Choosing (e_1 = red) forces (e_2 = green) since the binary constraint linking e_1 and e_2 specifies that they must have different colours. This, in turn, forces (e_3 = red), since the value of e_3 must be different to the value of e_2. The value of e_1 must be different to that of e_3, and so (e_1 = green), but we have already set (e_1 = red). Similar inconsistencies arise when starting by choosing (e_1 = green); the problem is clearly unsatisfiable.

And yet the constraint network in Figure 4.5 is arc consistent. Given that the edges in Figure 4.5 represent the binary predicate "is not the same colour as", the network is arc consistent (and trivially node consistent). For instance, red is a valid member of D_1 since there is a member of D_2 (i.e. green) which makes the constraining binary predicate true.

Given such uncertainties AI systems tend to use network consistency as a preprocess to a full search procedure, rather than as a substitute for the search. As we saw in the example of section 4.4, domain refinement can greatly reduce the search space for actual solutions in a very efficient manner. (We will return to the issue of time-complexity in the concluding section.)

Within the scope of this project we can make do with some of the uncertainty, but not all of it. In order to determine the felicity of a referring expression, we at least need to know whether or not it relates to any entities in the context. So we at least need to know whether or not a solution to the corresponding constraint problem exists; i.e. whether or not it is satisfiable.

But we do not require the actual set of tuples which satisfy the constraint problem. Here we can manage with a response less detailed than the actual set of solutions, but more informative than a simple yes/no decision on satisfiability. Over and above determining satisfiability, the procedure for reference evaluation must be capable of associating each variable in the constraint problem with a set of entities, such that every entity in each variable's set participates in some solution to the problem. In constraint network terms, we will say that an entity a in a domain D_i *participates* in some solution iff there exists a solution to the constraint problem in which the variable e_i is instantiated to a. This requirement of "full participation" is clearly stronger than satisfiability, but weaker than a need

for actual solutions.

So for the constraint problem

(4) $(\exists e_1,e_2)\ (e_1 \in D_1)\ (e_2 \in D_2)\ \text{man}(e_1)\ \&\ \text{visit}(e_1,e_2)\ \&\ \text{town}(e_2)$

the procedure must return two refined sets, D_1 and D_2, such that each element in D_1 is a man and visits a town in D_2, and each element in D_2 is a town and is visited by a man in D_1. We need not be concerned with how these sets' elements relate to each other in the space of actual solutions, since, as we will see in Chapter 5, the felicity of indefinite and definite noun phrases can be determined from these weaker results.

However, even though the requirement for actual solutions has been relaxed, we still require a decision on satisfaction, and, as we saw above, arc consistency alone does not provide such a decision. All it can do is rule out some values which definitely do not participate in a solution. Thus, at a first glance, it seems as though arc consistency is inadequate for our task and can only form the initial stage of a more complete search process.

But the interesting thing is this: in the example of section 4.4, node and arc consistency did everything we require of our reference evaluator. And in all the other instances of reference handled by the incremental processor of Chapter 5, node and arc consistency provide a sufficient guarantee that the reference problem is satisfiable and, moreover, that every entity in each domain of the consistent network participates in some solution.

Why are the limited, local operations of node and arc consistency adequate for the full noun phrases we address? Clearly the NPs must generate constraint problems which do not exploit the full complexity allowed in the general case, but why not? And why are node and arc consistency sufficient for their lower level of complexity? Is it just an accident?

Existing literature which applies network consistency techniques to linguistic problems rarely discusses these issues. The algorithms of Winston (1984), for word-sense disambiguation, Duffy (1986), for form-class disambiguation, and Rich, Wittenburg, Barnett and Wroblewski (1987; see also Rich, 1983), for reference evaluation, all rely on some variant of Waltz' (1975) original arc consistency algorithm, but their published work sheds little light on the above questions. However, Barton, Berwick and Ristad (1987; see also Barton, 1986) do relate similar questions to their use of arc consistency for morphological analysis, and conclude:

If it is true that natural-language problems have a special modular structure that

> distinguishes them from other logically possible problems, making them more like the easy problem than the hard problem [cf. complexity theory], constraint propagation represents an appropriate preliminary step toward processing methods that exploit that structure. (Barton, Berwick and Ristad, 1987, pp185-186)

Similarly, as we saw in Chapter 2, Mellish (1985) employs node and arc consistency techniques to resolve reference, and makes the following observations about the performance of the technique in his system:

> How important are [node, arc, and path] consistency in the reference evaluation task? Although we have not made detailed investigations, it is anticipated that merely considering node consistency will clear up reference ambiguities in almost all examples. However, the example
>
> A uniform rod ... is supported ... by a string ... attached to *its* ends.
>
> shows that arc consistency can also be needed. Whether examples can be found that require consideration of path consistency or even more stringent criteria remains to be seen. None have arisen to date. (Mellish, 1985, p58)

Interesting as these comments are, they do not answer the questions of the preceding paragraph (although Barton, Berwick and Ristad's suggestion is perhaps the most telling).

In actual fact, the apparent sufficiency of arc consistency for our own task is no mere accident. To see why, we must make an excursion into the comparatively recent but very significant results of Freuder (1982), not discussed by Mellish (1985) and Barton *et al* (1987). Freuder (1982) is interested in determining a state of affairs in which a search procedure will never have to backtrack over previous variable instantiations, and he shows that this state of affairs can be characterised in terms of the underlying connective structure of the constraint problem and the degree of consistency attained in a corresponding constraint network. The following section outlines Freuder's sufficient condition for backtrack-free search, and, then, by means of two corollaries shown to follow from Freuder's central theorem, we relate his result to the problem of reference. These two implications directly correspond to the above two requirements of our reference evaluator: that it can indicate both satisfiability and the full "participation" of each element of each domain in the consistent network.

Freuder's result will thus provide us with a precise, graph-theoretic connection between the structure of the constraint problem and the level of network consistency required for the problem. The subsequent section then applies this useful tool to the constraint problems determined by our full noun phrases, and, on the basis of another of Freuder's theorems, shows why node and arc consistency suffice. Importantly, this subsequent section will also show why another class of referring expressions (those involving bound-variable anaphora) requires more than node and arc consistency alone.

4.6.1. Freuder's Theorem and its Implications for Reference

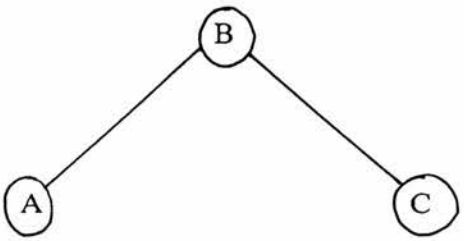
The present objective is to find out why the node and arc consistency of our reference-oriented constraint networks appears to guarantee the existence of a solution to the original CSP. As will emerge below, this endeavour seems closely related to Freuder's (1982) interest in determining a sufficient condition for backtrack-free search:[7] given a constraint satisfaction problem, is there some pre-determinable order of variable instantiation which guarantees that a search procedure will not have to backtrack and undo earlier variable instantiations at any point in the search? The following quotation from the original article should give a more precise picture of this concept and Freuder's general motivation for studying it:

> It would be desirable to have a more analytical understanding of the effort involved in a backtrack search. This effort is dependent on the structure of the problem and the order in which the search is conducted. The structure involves connective structure, the pattern of constraints among the variables, and contextual structure, the actual prohibition of various combinations of values. In terms of the usual "backtrack tree" picture, we may distinguish "vertical order," the order in which variables are chosen for instantiation, and "horizontal order," the order in which values are tested for a given variable.
>
> [Freuder's] paper analyses the relationship that holds between structure and order in the special case of "backtrack-free" search, where search effort, in terms of the amount of backtracking actually required, is minimal. "Backtrack-free" implies that once a value for a variable is chosen which satisfies constraints involving previously chosen values, the choice never has to be "unmade" because a dead-end is reached further down the tree. [Freuder] presents a relationship between the structure of the problem and the order of search which provides a sufficient condition for backtrack-free search... We say that a given vertical search order is *backtrack-free* if it guarantees a backtrack-free search regardless of the horizontal order of search. (Freuder, 1982, pp24-27)

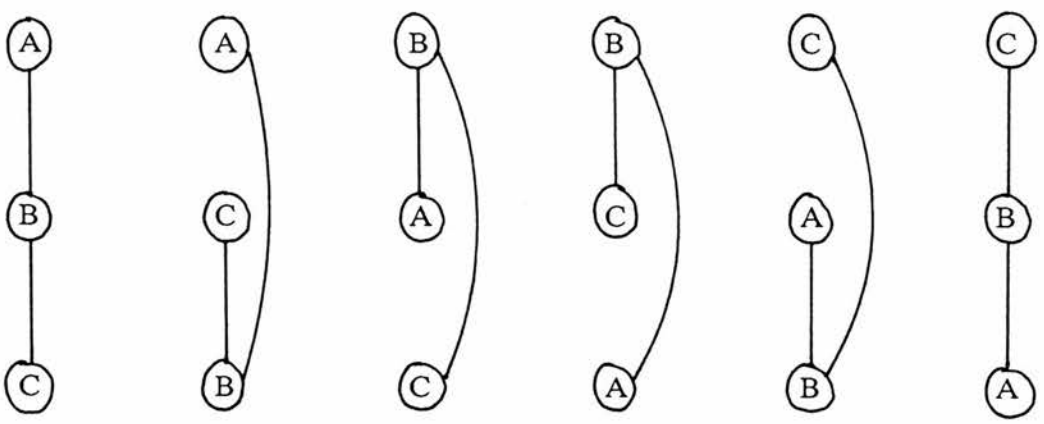
Freuder's sufficient condition for a backtrack-free vertical search order thus depends on both the connective, or "syntactic", structure of the constraint satisfaction problem, and the contextual, or "semantic", content of the problem. He goes on to characterise the syntactic parameter in terms of a graph-theoretic concept of "width", and the semantic parameter in terms of network consistency.

We will begin with the first of these. Freuder characterises the notion of width in terms of an undirected graph such as that of Figure 4.6(a), which represents a constraint problem with two binary constraints, on node pairs {A,B} and {B,C} respectively. In Freuder's terms, an *ordered constraint graph* is a constraint graph in which the nodes are arranged in a linear order. For example, the six different linear, vertical orderings for the

[7] See also Freuder (1985) for a related discussion of "backtrack-bounded" search.



(a) the constraint graph



(b) the ordered constraint graphs

(Adapted from Freuder, 1982, p27)

Figure 4.6. A constraint graph and its six different orderings.

graph of Figure 4.6(a) are shown in Figure 4.6(b); each ordering simply represents a different way of drawing the original constraint graph, and so preserves the original connective structure.

Freuder (1982, p26) then defines the *width of a node* in a linear, vertical ordering as the number of edges leading from that node to previous nodes above it in the ordering. The *width of an ordering* is defined as the maximum node width of the ordered graph. And the *width of a constraint graph* is simply the minimum width of all the orderings of the constraint graph.[8]

Consider the graphs in Figure 4.6 in the light of these definitions. The width of node

B in the first ordered graph is 1, whereas the width of B in the second ordering is 2. The width of the first ordered graph is 1, and the width of the second ordering is 2. Finally, the width of the original constraint graph is 1, the minimum of the widths of the six ordered graphs.

The intuition behind this concept is that a minimal-width ordering of a constraint graph corresponds to the vertical order of search for the associated constraint problem. By instantiating variables in the order indicated by the minimal-width ordered graph (reading from top to bottom), we can be sure that when the variables are instantiated they will directly depend on no more previous instantiations than would have been the case with another ordering.

Of course, the order of the variables is just one part of Freuder's condition for backtrack-free search. We also have to know about the state of consistency in the value domains. Therefore, let us now turn to Freuder's method for determining the semantic content of a CSP, a form of network consistency called "*k*-consistency". *K*-consistency is a generalisation of Mackworth's (1977a) notions of node, arc, and path consistency. Recall that node consistency involves considering single nodes in isolation and checking to see if their domains satisfy their constraining unary predicate. Arc consistency, however, considers two nodes at once; checking that all the elements in one node's domain are consistent with some value in the domain of an adjacent node linked directly by a binary constraint on the two nodes. Similarly, path consistency can consider three nodes at once, checking that all the values in two adjacent nodes' domains are simultaneously consistent with some value in the domain of a third node which lies on a path between the two adjacent nodes.

Recognising this progression in Mackworth's three levels of consistency, Freuder (1978) introduces the concept of *k*-consistency for an *n*-variable constraint network, where $k \leq n$. A constraint network is *k-consistent* iff given any instantiation of any k - 1 variables such that the constraints directly among those variables are satisfied, it is possible to find an instantiation of any *k*th variable such that the k values taken together satisfy the constraints directly among the k variables. Thus node, arc, and path consistency correspond to *k*-consistency where $k = 1, 2$, and 3, respectively. Take arc consistency, for instance, which corresponds to 2-consistency: given any instantiation of any one variable which satisfies its constraining unary predicate, it should be possible to find an instantiation of any second variable such that the two values together satisfy the constraining binary predicate. (If

[8] Note that unary constraints do not affect the width of a graph and so are irrelevant as far as Freuder is concerned.

there is no such direct binary constraint, then the two variables are linked by a "non-constraint"; see Freuder (1978).)

Freuder (1982, p26) reviews these notions and defines j-consistency for all $j \leq k$ as *strong k-consistency*. Thus, whereas 2-consistency corresponds to arc consistency, strong 2-consistency implies both node and arc consistency.

We are now in a position to consider Freuder's characterisation of a state of affairs which guarantees backtrack-free search. As the following theorem and proof (from Freuder, 1982, p27) shows, this situation is captured in terms of the above notions of connective structure (width), contextual structure (the level of consistency attained), and vertical order:

THEOREM 1. (Freuder) *Given a constraint satisfaction problem:*

(1) *A vertical search order is backtrack-free if the level of strong consistency is greater than the width of the corresponding ordered constraint graph.*

(2) *There exists a backtrack-free vertical search order for the problem if the level of strong consistency is greater than the width of the constraint graph.*

PROOF. In instantiating any variable v we must check consistency requirements involving at most j other variables, where j is the width of the node. If the consistency level k is at least $j + 1$, then given prior choices for the j variables, consistent among themselves, there exists a value for v consistent with the prior choices. □

Let us now see how this result relates to the problem of determining felicitous noun phrase reference. With what was said before this subsection very much in mind, it is clear that Part Two of Freuder's theorem is more relevant than Part One for our present purposes. Part One tells us which of the ordered constraint graphs for a constraint problem will guarantee a backtrack-free vertical search order. But we are not so interested in an efficient means of searching the domains of a strongly consistent network for an actual solution; first and foremost we are interested in whether a solution exists at all. Thus, Part Two is more relevant: that we can be sure a backtrack-free vertical search order exists if the level of strong consistency attained in the network is greater than its width.

Freuder (1982) is not explicit about the relationship of his result to the question of satisfiability. However, Part Two of Theorem 1, together with its proof, clearly shows that if the level of strong consistency k is greater than the width of the constraint graph j, then the constraint problem is satisfiable. Consider instantiating the variables in an order corresponding to the top-to-bottom order of nodes in the minimal width ordered graph. In

circumstances where $k > j$, we know that at each node we can make an instantiation from that node's domain which is consistent with the previous instantiations. And since we can do this for every node in the ordered graph, from the first to the last node, there must be a solution. Given the relevance of this result to the question of felicitous noun phrase reference, it seems appropriate to enshrine it as a trivial corollary of Freuder's theorem, Corollary 1 below. Of course, satisfiability is only one necessary aspect of the felicity of a noun phrase. We also require a stronger result, namely a guarantee that every element in each domain of a consistent network participates in some solution to the problem. Although Freuder does not mention it, this also follows from his theorem. I therefore state the following two corollaries of Freuder's work, to connect it with the reference task:

COROLLARY 1. *A constraint satisfaction problem is satisfiable if the level of strong consistency is greater than the width of the constraint graph.*

COROLLARY 2. *Every element in each domain of a constraint network participates in some solution to the problem if the level of strong consistency is greater than the width of the constraint graph.*

While the first of these corollaries simply re-phrases an aspect of Theorem 1, the second may be less obvious and requires elaboration. Let us take it in stages, first considering the case where strong 2-consistency has been attained in a constraint network of width 1. A minimal-width ordering for such a network with three nodes {A,B,C} is shown in Figure 4.7(a), along with the schematic domains of each node. Assume that the network contains no unary constraints, and so is trivially node consistent. A binary constraint P links nodes A and B, and a binary constraint Q links nodes B and C. Given that the network is strongly 2-consistent, we know that (12)-(15) are true:

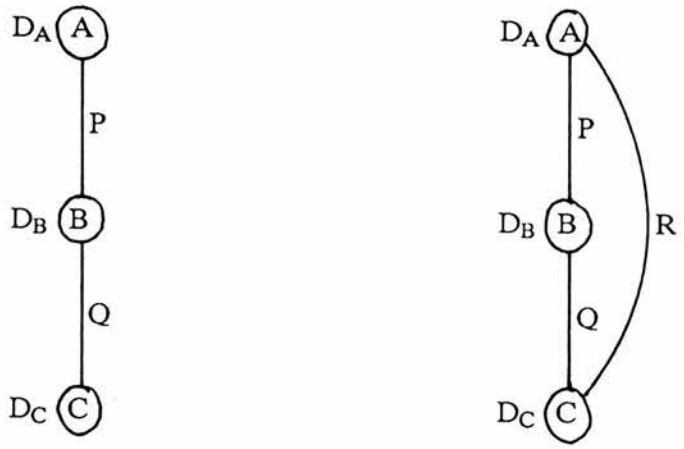
(12) $(\forall a \in D_A)\ (\exists b \in D_B)\ P(a,b)$

(13) $(\forall b \in D_B)\ (\exists a \in D_A)\ P(a,b)$

(14) $(\forall b \in D_B)\ (\exists c \in D_C)\ Q(b,c)$

(15) $(\forall c \in D_C)\ (\exists b \in D_B)\ Q(b,c)$

From Theorem 1 we know that a backtrack-free search exists. From the definition of "backtrack-free search", this means we can start with any $a \in D_A$ and be certain of finding some $b \in D_B$ consistent with P(a,b). But since both (12) and (13) hold, we can consistently reach any $b \in D_B$. So not only does every element of D_A participate in a solution, so does every element of D_B. By the same reasoning we can see that every element of D_C will participate in a solution, and, if the ordered network were longer, so would all the other domains' elements.



(a) ordered network of width 1 (b) ordered network of width 2

Figure 4.7. Ordered networks of widths 1 and 2.

Now take the case of Figure 4.7(b), an ordered network of width 2 which we assume to be strongly 3-consistent. So whereas Figure 4.7(a) was node and arc consistent, in addition all paths of length 2 in 7(b) are path consistent. Formally, then, the following are true in addition to (12)-(15):

(16) $(\forall a \in D_A)\,(\forall b \in D_B)\,P(a,b) \supset (\exists c \in D_C)\,Q(b,c)\ \&\ R(a,c)$

(17) $(\forall b \in D_B)\,(\forall c \in D_C)\,Q(b,c) \supset (\exists a \in D_A)\,P(a,b)\ \&\ R(a,c)$

(18) $(\forall a \in D_A)\,(\forall c \in D_C)\,R(a,c) \supset (\exists b \in D_B)\,P(a,b)\ \&\ Q(b,c)$

From the argument above, the search procedure can reach any $b \in D_B$. Because the search is backtrack-free (i.e. because (16) holds), the search procedure can reach some $c \in D_C$ irrespective of which $a \in D_A$ and $b \in D_B$, such that P(a,b), are chosen. Moreover, because of (17) and (18), all $c \in D_C$ are valid continuations; therefore every element in D_C participates in a solution.

For the general case, consider a network of width j which is strongly $(j + 1)$-consistent. We wish to show that every element in the domain of some node N, D_N, participates in a solution. Supposing that the width of node N is j, let us denote the set of these j nodes, which directly constrain N from above in the vertical ordering, as S_j. From Theorem 1, we know that node N can be consistently instantiated with respect to the nodes in S_j. So, from Theorem 1, there exists some value in D_N which participates in a solution.

To see that *every* value participates, take any subset $S_{j-1} \subset S_j$ such that $S_{j-1} = S_j - \{M\}$, for some node $M \in S_j$ (i.e. $|S_{j-1}| = j - 1$). Since the network is $(j + 1)$-consistent, and $|S_{j-1} \cup \{N\}| = j$, we can make any instantiation of the nodes $S_{j-1} \cup \{N\}$ which is consistent among themselves, and always find a value for M in its domain such that all the values taken together satisfy the constraints directly among the set of $j + 1$ nodes, $S_{j-1} \cup \{N\} \cup \{M\}$. This means that for all the $n \in D_N$, there exists an instantiation of the nodes in S_j. Thus, every value in D_N represents a valid continuation of the search for some consistent instantiation of the nodes S_j, and since the search is backtrack-free, must participate in some solution to the problem.

This completes our discussion of Freuder's theorem and its relationship to the proposed use of constraint networks. In short, a constraint network adequately characterises the extensions of a phrase when the level of strong consistency attained in that network is greater than its width. And generally speaking, the more inter-connections in the network, the greater its width. On the basis of these observations and another of Freuder's results, we can now go on to see why node and arc consistency are adequate for the current application.

4.6.2. Examples: Full Noun Phrases and Bound-Variable Anaphora

Why do the constraint problems determined by our full noun phrases consistently fail to exploit the degree of complexity allowed by the general constraint problem, and why is strong 2-consistency adequate for their lower level of complexity? The answers lie in the following theorem of Freuder (1982, p30):

THEOREM 2. (Freuder) *A connected constraint graph (with more than one node) has width 1 iff it is a tree. More generally, a constraint graph has width $\leq$ 1 iff it is a forest.*[9]

PROOF. See Freuder (1982, p30).

All of the full noun phrases addressed in this thesis derive constraint networks in the form of trees. Their logical forms do not yield graphs with cycles, and it is this common acyclic property which produces a less complex graph than allowed by the general, cyclic case. Since they yield trees, from Theorem 2 we know their width must be 1. And since they all have width 1, our two corollaries to Theorem 1 guarantee that strong 2-consistency (i.e. node and arc consistency) will be adequate to compute their extensions.

---

[9] A graph is *connected* if for every pair (i,j) of distinct vertices there is a path from i to j. A graph without any cycles is a *forest*, and a *tree* is a connected forest. In other words, a forest is a

For example, consider again the undirected constraint network of Figure 4.1, which represents the NP *a man who visits a town on a river*. The network is acyclic. (Although the unary predicates are drawn as loops, and so might seem to introduce cycles, this is simply a notational convenience and does not affect the essential structure — and therefore width — of the network; see Footnote 2.) By observation, all of the noun phrases we deal with in the next chapter derive such tree-structured networks; none produce logical forms in which the variables are bound in a cyclic fashion.

It is worth clarifying the status of this result, for in Chapter 5 we define a system of semantic translation which certainly does have the power to derive cyclic logical forms. The syntactic-semantic framework of section 5.1 allows cycles to be introduced in at least two distinct ways. First, our definition of a lexical entry allows each word to be associated with an arbitrary conjunction of constraints (a "constraint list"), and this may itself have a cyclic structure. The second source of cycles comes from the combinatory rules, which have the role of combining two disjoint "daughter" constraint lists into a single "mother" constraint list. Even if we assume that each of the daughter constraint lists is tree-structured, the semantic component of a rule such as function application may potentially create cycles in the composite, mother constraint list. Thus it is not possible to guarantee that our general system of translation will always derive a tree-structured logical form for an input noun phrase, regardless of the lexicon.

Nevertheless, the above interpretation of Freuder's work explains why arc consistency is adequate for our example noun phrases, and furthermore tells us that it will be always be adequate for these expressions, whatever the state of the referential context. And although our general mechanism for semantic translation can derive cyclic structures, it is quite conceivable that the specific fragment of lexicon and rules provided in Appendix A.2 is indeed limited to the generation of tree-structured logical forms. However, whether or not this is true is left as a matter for further research.

At this point, the principal aims of this chapter have been fulfilled. We have seen how to characterise noun phrase reference as a constraint satisfaction problem, and have been introduced to a particular algorithm for computing that reference, an algorithm for determining node and arc consistency in the corresponding constraint network. Importantly, Freuder's work has provided a basis for demonstrating that the limited operations of node and arc consistency are adequate for the noun phrases we address, whatever the state of the referential context.

However, having introduced some technical machinery to confront one set of referring expressions, it seems interesting to apply it to some other cases, even if we do not show how they could be incrementally processed in the remainder of the thesis. It is particularly interesting to ask if there are any classes of expressions which require more than node and arc consistency alone.

The short answer is that there certainly are such classes. Here we will focus on complex noun phrases like (19) which involve an embedded pronoun:

(19) a man who visits a town near his birth-place

On the assumption that (19) refers non-specifically to some man known to the hearer, and depending on the state of the discourse, the possessive pronoun *his* may refer either to some male entity salient in the hearer's discourse model or be bound to the extension of the entire complex noun phrase in which it is embedded. In the latter case it thus becomes an instance of what Partee (1978), Reinhart (1983) and others have called "bound-variable anaphora". To see why, consider the following CSP-style logical form for the bound-variable interpretation of (19):

(20) $(\exists e_1,e_2,e_3)\ (e_1 \in D_1)\ (e_2 \in D_2)\ (e_3 \in D_3)$
$\text{man}(e_1)\ \&\ \text{visit}(e_1,e_2)\ \&\ \text{town}(e_2)\ \&\ \text{near}(e_2,e_3)\ \&\ \text{birthplace}(e_3,e_1)$

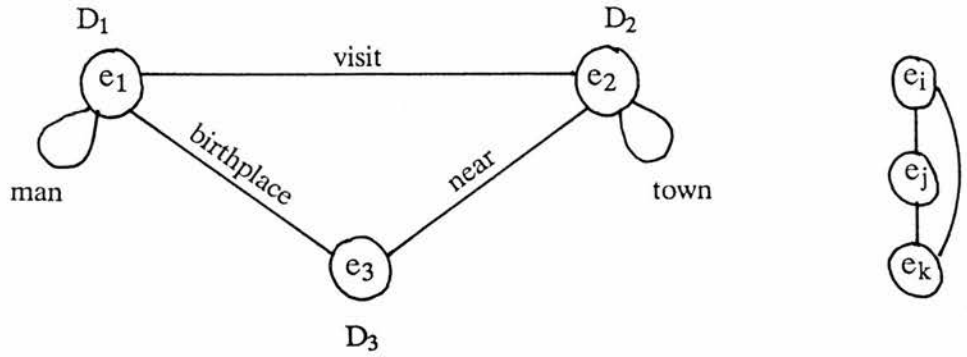
That is, a man e_1 visits a town e_2 which is near the birthplace e_3 of the man e_1, whoever he is; the variable representing the extension of the *male person whose birthplace it is* has been bound to the variable representing the extension of *a man who visits ....* It is this bound-variable interpretation of (19) which we will now consider in terms of an underlying evaluation engine based on network consistency.

To determine the level of strong consistency required to compute extensions for the expression, we must determine the width of its constraint graph. The constraint graph corresponding to (20) is shown in Figure 4.8(a).[10] Unlike our full noun phrases, which yield trees, this graph contains circuits, and so Theorem 2 does not help us find its width. Instead consider the six different orderings for Figure 4.8(a). These are all of the same schematic form indicated by the ordered graph in Figure 4.8(b); all have width 2, and thus the width of the constraint graph in Figure 4.8(a) is 2. Given that we must attain a level of strong consistency greater than this width, strong 3-consistency might be required to determine the felicity of (19) in some contexts.

---

disjoint union of trees. (Bollobas, 1979)

[10] Note that the edges representing the predicates *birthplace* and *near* have been labelled somewhat informally in Figure 4.8(a) in that e_3 is the birthplace of e_1, and not the opposite, as the



(a) constraint graph (b) ordering

Figure 4.8. Constraint graph and ordered constraint graph for (20)

It is not difficult to think of such a context. Suppose we have a situation involving two men, two towns, and two birth-places, related to each other as follows:

(21) { man(man1), man(man2), town(town1), town(town2),
visit(man1,town1), visit(man2,town2),
near(town1,place2), near(town2,place1),
birthplace(place1,man1), birthplace(place2,man2)}

Careful inspection of (21) will reveal that no man visits a town near his *own* birth-place. The first man *man1* visits *town1*, but *town1* is near *place2*, the second man's birth-place, not *place1*. Similarly, *man2* visits *town2*, which is near the first man's birth-place, but not his own.

Arc consistency will do nothing for us here. Given the problem in (20) in the context of (21), a node and arc consistency procedure such as that discussed above will return a consistent network with the domains set as:

D_1 = {man1,man2}
D_2 = {town1,town2}
D_3 = {place1,place2}

And yet, in the context, the constraint problem is unsatisfiable. The arc consistency procedure cannot see this because the unsatisfiability of the problem stems from a global dependency introduced by the bound pronoun; the pronoun allows the variable e_1, representing the head noun, to function also as an argument to the predicate *birthplace*.

figure might suggest.

As far as arc consistency is concerned, all is well: every man in D_1 visits a town in D_2, and every town in D_2 is visited by a man in D_1; every town in D_2 is near a place in D_3, and every place in D_3 has a town in D_2 near it; every place in D_3 is the birth-place of some man in D_1, and every man in D_1 has a birth-place in D_3. Because of the local nature of the consistency checks, and the "narrow bandwidth" of the domains as the only means of communication between connected arcs, the arc consistency procedure does not spot that somewhere along the way the wires have been crossed.

Similarly problematic contexts can be constructed in which a unique solution to the reference problem exists but where node and arc consistency fail to rule out some invalid candidates, leaving domains containing more than one element.

However, although the network of Figure 4.8(a) is node and arc consistent, it is not path consistent (i.e. 3-consistent). Consider the path (e_2,e_1,e_3), which is path consistent iff

$(\forall x \in D_2)\ (\forall z \in D_3)\ \text{near}(x,z) \supset (\exists y \in D_1)\ \text{visit}(y,x)\ \&\ \text{birthplace}(z,y)$

While the instantiations {x = town1, z = place2} and {x = town2, z = place1} satisfy the binary constraint near(x,z), for neither of these instantiations does there exist a $y \in D_1$ such that *visit(y,x)* and *birthplace(z,y)* both hold. Hence this path, and in fact all the others in Figure 4.8(a), are inconsistent in the context of (21). Accordingly, Mackworth's (1977a) algorithms for attaining path consistency would detect the unsatisfiability and report failure for this network, as would Freuder's (1978) algorithm for "synthesising" constraints when taken to consistency-level 3.

Actually, it is not just pronouns which produce a more complex constraint problem. Cycles can also occur in the graph when syntactic analysis induces a certain kind of "double binding" in the logical form. Consider the following noun phrase, for example, containing what Engdahl (1983) calls a "parasitic gap".

(22) an apple which a man ate without peeling

Assuming a first-order treatment of adverbial modification in the spirit of Davidson (1967) and Hobbs (1985), which will be explained more fully in the next chapter, we can add a third variable to the predicate *eat* to represent the event of eating. Ignoring the tense of the expression, (22) translates into a logical form in the shape of the following CSP:

(23) $(\exists\ e_1,e_2,e_3)\ (e_1 \in D_1)\ (e_2 \in D_2)\ (e_3 \in D_3)$
$\text{event}(e_1)\ \&\ \text{man}(e_2)\ \&\ \text{apple}(e_3)\ \&\ \text{eat}(e_1,e_2,e_3)\ \&\ \text{withoutpeeling}(e_1,e_3)$

So there exists an event e_1 in which a man e_2 eats an apple e_3 and, furthermore, the event e_1 is carried out without peeling the apple e_3. Now take a context akin to (21), say one

involving two eating events, each with a different man and apple:

(24) {event(i1), event(i2), man(man1), man(man2), apple(apple1), apple(apple2),
eat(i1,man1,apple1), eat(i2,man2,apple2),
withoutpeeling(i1,apple2), withoutpeeling(i2,apple1)}

Here, there is no apple *a* such that a man ate *a* without peeling *a*. Rather, *man1* ate *apple1* without peeling the other man's apple, *apple2*; and vice versa. And, in this context, Mackworth's (1977b) algorithm for arc consistency in networks of n-ary constraints would fail to determine the unsatisfiability. If we were to treat phrases with such double, parasitic gaps, which we do not,[11] higher levels of network consistency would again be required.

4.6.3. Conclusion

Researchers have often adopted an empirical approach to the usefulness of network consistency for their task in hand (e.g. the empirical observations of Mellish (1985, p58), quoted above). However, by carefully analysing the exact requirements of a procedure for reference evaluation, and relating these requirements to Freuder's conditions for backtrack-free search, we have seen that node and arc consistency will always meet these requirements for our particular task. We have thus significantly closed the empirical gap between linguistic expressions (or, at least, their logical forms) and network consistency.

These connections between the reference problem and existing results in constraint satisfaction have implications beyond the scope of this project. While our own interest in network consistency is with its suitability for incremental processing, other projects may be more interested in its general efficiency. Reference evaluation and other linguistic tasks are often phrased as CSPs, and yet, as Mackworth and Freuder (1985) point out, the general constraint satisfaction problem is an NP-complete problem.[12] Along with the many other problems in the same NP-complete class, it is believed to be inherently intractable, apparently solvable only in exponential time. For example, in the worst case, a backtrack search takes time exponential in the number of variables (Mackworth and Freuder, 1985).

When *k* can be limited, consistency algorithms are far more efficient and can drastically reduce the value domains for each variable before a full search procedure (such as backtracking) takes over. Moreover, it is likely that we can do without post-processing by a full search method for many applications involving noun phrase reference, as is the case

[11] Sentences with parasitic gaps are not handled because we do not include Steedman's rule corresponding to the S-combinator in our combinatory grammar. (Cf. Steedman, 1987b.)

[12] For an introduction to complexity theory and NP-completeness, see Garey and Johnson

for the present project. For noun phrases which yield tree-structured constraint networks, the worst-case time-complexity for obtaining node and arc consistency is in the order $O(d^3n)$, where d is the size of each initial value domain and n is the number of variables in the problem (Mackworth and Freuder, 1985). This means that full noun phrases of the kind implemented here can be evaluated in time *linear* with the number of variables in the corresponding CSP, ignoring any syntactic ambiguity. Even the expressions involving bound-variable anaphora can be evaluated in time cubic with the number of variables, since the time-complexity of path consistency is the polynomial order $O(d^5n^3)$ (Mackworth and Freuder, 1985). Although slower than arc consistency, in the general case this is more efficient than backtracking.

Of course, we must first determine the width of a constraint network before invoking consistency operations, so that we know which level of strong consistency to aim for. Whereas this was done earlier by considering all $n!$ vertical orderings, Freuder (1982, p28) demonstrates a more efficient technique for determining width in terms of "maximal subgraph linkage". His algorithm appears to run in polynomial-time, in the worst case $O(n^2)$. In principal, this opens up the prospect of an efficient reference evaluator which knows just how much effort to put into each expression it is given: a program which first determines the width j of the constraint graph for the expression, and then achieves strong $(j + 1)$-consistency in the network. However, the possible speed-up to be gained from such a two-stage approach will, almost certainly, depend on the classes of referring expression found in the application in question, and so we will not pursue this line of thinking here. (Freuder himself (1982, p32) is rather negative about the practical applications of the notion of width.) Needless to say, as we have seen here, the concept of width still provides an extremely useful tool in the design of the overall system. And, indirectly, this is very likely to obtain improvements in the speed of reference evaluation.

4.7. Summary

There have been two points to this chapter. The first has been to introduce the basic process for reference evaluation. This involved specifying the reference task as a classical constraint satisfaction problem, and then showing how this specification makes the problem amenable to network consistency methods, among other solution schemes. The particular algorithm presented above (from Mackworth) maintains node and arc consistency in a

(1979). For some applications to linguistic problems, see Barton, Berwick and Ristad (1987).

network of n-ary constraints, and uses constraint propagation to explore the effects of a change in the state of the network. Although this algorithm is not in precisely the form we require for incremental reference evaluation (since it assumes the entire reference problem as input), adapting it for our purposes is a relatively simple matter, as the next chapter shows.

The second main purpose of this chapter has been to show why this algorithm is powerful enough for the full noun phrases addressed by the thesis. By considering the exact requirements of our reference evaluator, and in particular recognising that we do not need to enumerate complete solutions to reference-based constraint problems, we were able to utilise Freuder's results linking the adequacy of network consistency to the structure of the constraint network. This allowed us to observe that because our full noun phrases always determine tree-structured constraint networks, node and arc consistency will always be powerful enough to accurately compute their extensions, whatever the state of the referential context. Significantly, it also enabled us to see that certain referring expressions will require more than node and arc consistency alone — for example, those involving bound-variable anaphora.

Chapter 5

A Model of Incremental Reference Evaluation

The two preceding chapters have described our two computational starting points for incremental interpretation: combinatory categorial grammar and network consistency. The present chapter puts these systems together, and goes on to produce a model of definite reference evaluation in the context.

The chapter is organised as follows. Section 5.1 introduces a simple method for associating semantic translations with syntactic categories in categorial grammar. This translation scheme has the effect of associating each constituent of the grammar with a miniature constraint satisfaction problem, so that the extensions of each constituent may be readily computed by network consistency techniques. As section 5.2 shows, this produces a model of reference resolution in which extensions are gradually resolved as a noun phrase is parsed from left to right. Section 5.3 turns to the question of definite reference, and formalises the semantics of the definite article as a constraint of uniqueness, interpreted as a meta-level check on the results of domain refinement. Finally, section 5.4 demonstrates the usefulness of this approach to semantic interpretation by confronting the two problem areas discussed in Chapter 1: the problematic rabbit phrase and the problem of resolving local syntactic ambiguities.

5.1. Semantic Translation

This section undertakes the first step in connecting the syntactic and referential aspects of the two preceding chapters, by introducing a scheme for translating syntactic units into intermediate semantic representations. The general strategy will be to keep the semantic translations simple, so that they may be readily interpreted as constraint satisfaction problems of the kind seen in Chapter 4. With this in mind, we translate each constituent into a "flat" conjunction of first-order predicates. Although this might be a rather restricted form of semantic representation, it will be convenient in approaching the central issue of incremental reference evaluation. As with the method of semantic translation illustrated in Chapter 3, the scheme works by associating each word in the lexicon with a syntactic category and a corresponding semantic representation. The combinatory rules are

then augmented with an appropriate means of semantic combination, which makes limited use of unification instead of the lambda-conversion seen in Chapter 3. The approach thus follows the systems of Gawron *et al* (1982), Rosenchein and Shieber (1982) and Schubert and Pelletier (1982) in assuming an essentially compositional style of translation.

Let us start with the lexical definition of a noun. Suppose we associate a constraint with each noun and a semantic variable with its category, so that the lexicon characterises the noun *man* as in (1).

(1) man := N_{e_2} : [man(e_2)]

This defines *man* as an N linked to the semantic variable e_2. (The identity of this variable is quite arbitrary.) The definition also associates a constraint with the noun, which, in this case, indicates that the domain of e_2 should consist only of men.

Function categories differ only in that they are coupled to *mappings* between variables. An intersective adjective like *one-legged*, a mapping from nouns to nouns, appears as:

(2) one-legged := N_{e_1}/N_{e_1} : [one-legged(e_1)]

(2) states that the variable associated with the argument noun is the same as the variable associated with the modified, result noun. The definition introduces the single constraint *one-legged(e_1)* which, like *man(e_2)*, involves a predicate over entities in the context.

These examples illustrate the basic form of each definition in the lexicon: a word is associated with a CCG category, augmented with semantic variables, and some constraint information. And on the basis of these examples, we can observe some general characteristics true of all constituents. For instance, all constituents — lexical or otherwise — will be associated with an augmented category like those in (1) or (2). More precisely, all categories combine syntactic and semantic information in accordance with the following definition:

(3) Categories

a. S, S', NP and N are *syntactic categories*.

b. e_1, e_2, ..., e_i are *semantic variables*, where *i* is an integer.

c. If C is a syntactic category and v is a semantic variable, C_v is a *category*.

d. If A and B are categories, A/B is a *category*.

e. If A and B are categories, A\B is a *category*.

The definitions in (1) and (2) also exemplify the general form of the constraint information associated with any constituent. The constraints are always represented in a list,

since a word may introduce a number of constraints, or none at all. The list is written by enclosing square brackets around the constraints, as in the programming language Prolog; so (1) involves a list of one constraint. Some assumptions underlie this part of a constituent's translation. First, a list of constraints is always interpreted as a *conjunction* of constraints. Second, the constraints relating to a constituent can be thought of as its *sense*: unlike the identity of the variables or the particular values they may assume, the constraints remain the same in all contexts. A third issue concerns the variables occuring in a constraint list: these will be seen as *existentially quantified* by the consistency algorithm in charge of evaluating their extensions.

Let us now move on to the combinatory component of the grammar, and consider the role of the combinatory rules in linking these translations. The combinatory rules have particular responsibility for equating semantic variables. For example, if the above adjective (2) and noun (1) are to be combined by forward application, the variables e_1 and e_2 should be equated, because we will want them to represent the same extension.

A simple way of doing this is to reformulate the combinatory rules so that the usual syntactic cancellation is subsumed under a more general operation of unification, which will simultaneously identify semantic variables and cancel syntactic categories. Specifically, the rules are re-phrased as operations of *term unification* (as found in Prolog), which Pereira and Shieber (1987) and others have shown to be generally useful in adducing semantic translations. Here the rules of application and composition will unify terms of the form C_v, each term thus consisting of a syntactic category C and a semantic variable v. At the same time, the rules will concatenate the constraint lists of the reduced constituents.[1]

Function application and composition are therefore reformulated as (4) and (5) below, where the symbol "+" is used to signify list concatenation, and the unification of category terms is expressed by the sharing of variables within each rule.

(4) Forward and Backward Application

a. X/Y : T1 Y : T2 ==> X : (T1 + T2)
b. Y : T1 X\Y : T2 ==> X : (T1 + T2)

[1] Note that Calder, Klein and Zeevat's (1988) Unification Categorial Grammar also uses term unification for semantic translation, though the form of their syntactic/semantic representations is somewhat different to ours.

(5) Forward, Generalised Forward, and Crossed Backward Composition

a. X/Y : T1 Y/Z : T2 ==> X/Z : (T1 + T2)
b. X/Y : T1 (Y/W)/Z : T2 ==> (X/W)/Z : (T1 + T2)
c. Y/Z : T1 X\Y : T2 ==> X/Z : (T1 + T2)

Here, the variables X, Y and Z range over *categories*: syntactic categories subscripted by semantic variables. The operation of unification simultaneously manipulates the syntactic and semantic parts of a category. By treating syntactic categories as constants, syntactic cancellation is achieved by unifying syntactic categories. At the same time, the corresponding subscripts are unified: this way, pairs of semantic variables are bound together, becoming the same variable. Typically, this means that the component constraint lists T1 and T2 will now share variables, so that the concatenation (T1 + T2), which is simply a notation for conjunction, constrains the variables further.

We will come across a number of instances of rule applications in the course of this section. For the meantime, consider the forward application of the adjective *one-legged*, defined in (2), to the noun *man*, defined in (1). Fitting the definitions into the first and second components of the left-hand side of (4a) causes the following unifications:

$X = N_{e_1}$
$Y = N_{e_1} = N_{e_2}$
$T1 = [\text{one-legged}(e_1)]$
$T2 = [\text{man}(e_2)]$

Syntactic cancellation and the identification of semantic variables results from the simultaneous instantiations $Y = N_{e_1}$ and $Y = N_{e_2}$; N_{e_1} unifies with N_{e_2} because N unifies with N, and e_1 unifies with e_2. Assuming that e_1 and e_2 unify as e_1, *man* is now predicated over e_1, the variable associated with the modified noun:

(6) one-legged-man

$N_{e_1} : [\text{one-legged}(e_1),\text{man}(e_1)]$

The semantics of other functions follows straightforwardly.[2] Consider type-raised functions, for example. Recall from Chapter 3 that nominal type-raising allows a noun to be regarded as function over noun-modifying functions into Ns. In the present terms, the rule of nominal type-raising can be re-stated as:

(7) Nominal Type-Raising

$N_v : T \uparrow N_v/(N_v\backslash N_v) : T$

Here, v is a variable over semantic variables e_i. Because this rule raises a noun over a

[2] The terms *function* and *argument* are now used to describe mainly the syntactic aspect of a constituent: the constraints in a constraint list are essentially "relational", and do not bear a

modifying function, we can assume that each N in (7) corresponds to the same semantic variable v. And, according to (7), the constraint list for a type-raised noun is just as it is for an atomic noun: as in Steedman's canonical version of CCG, type-raising does not alter the semantics of a constituent.

Thus applying the rule of (7) to the noun *man* in (1) yields the following alternative definition:

(8) man := $N_{e_2}/(N_{e_2}\backslash N_{e_2})$: [man(e_2)]

Note that the only difference between (1) and (8) is the noun's syntactic status. In particular, type-raising allows the noun to compose with *in*, categorised as (N\N)/NP. Such prepositions are syntactic functions taking an NP argument on their right to form another function, which must modify a noun on the left. Because of its relational standing, *in* maps between two distinct semantic variables:

(9) in := $(N_{e_1}\backslash N_{e_1})/NP_{e_2}$: [in(e_1,e_2)]

Thus, e_1 will be identified with the variable of the noun and related, by the constraint, to e_2, which represents the prepositional object.

These lexical representations are illustrated by the following incremental analysis of the fragment *one-legged man in*. *One-legged* and *man*, as defined in (8), can combine by forward composition (5a), to yield the representation in (10a) for *one-legged man*. This composite representation goes on to compose with the preposition, defined in (9), unifying the appropriate variables and concatenating the component translations. The result is (10b). (Note that variable names are seen as local to each entry, and so although both (10a) and (9) include a variable named e_1, these are in fact distinct variables until they are unified.)

(10) a. one-legged-man

$N_{e_1}/(N_{e_1}\backslash N_{e_1})$: [one-legged(e_1),man(e_1)]

b. one-legged-man-in

N_{e_1}/NP_{e_2} : [one-legged(e_1),man(e_1),in(e_1,e_2)]

Verbs introduce relations akin to those of prepositions. Here, however, it is convenient to add an extra variable to the constraint, to allow a straightforward account of verb-modification. So the transitive verb *ate* specifies a 3-place relation between the agent, the object, and the event of the agent eating the object. In the following definition of (11), these are e_2, e_3 and e_1 respectively (as in all such examples, the tense of the

expression is ignored in the semantics).

(11) ate := $(S_{e_1}\backslash NP_{e_2})/NP_{e_3} : [eat(e_1,e_2,e_3)]$

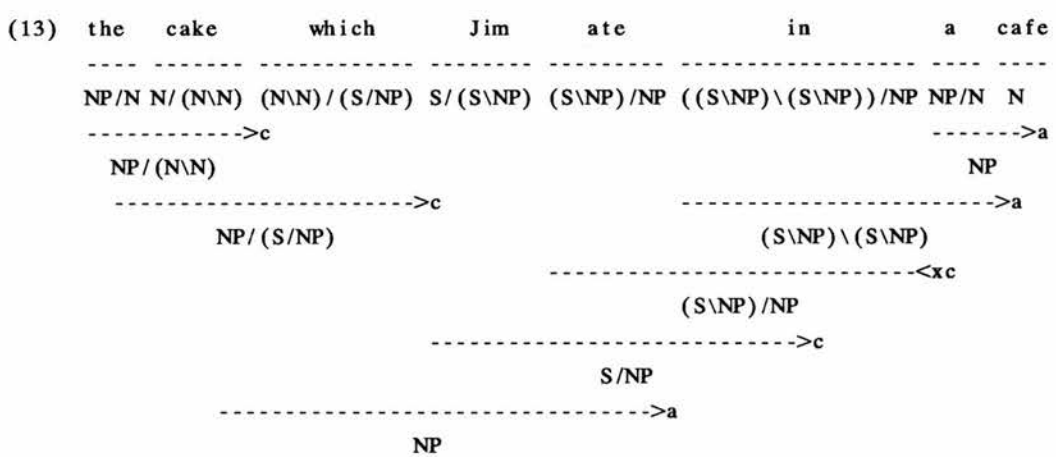
The verb is a syntactic function from an NP object to another function which takes an NP subject and returns a sentence. Notice that the third "event" variable, e_1, which we will generally refer to as the *index* of the constraint, is associated with the S category.

A second category for *in*, for instance, specifies verbal modification:

(12) in := $((S_{e_1}\backslash NP_{e_2})\backslash(S_{e_1}\backslash NP_{e_2}))/NP_{e_3} : [in(e_1,e_3)]$

Here, *in* is a syntactic modifier of verbal phrases bearing the category S\NP. For our purposes, it is adequate to treat the modifier semantically as an S-modifier; so the predicate applies to e_1, which associates with S.[3]

By way of an example, consider the NP analysis in (13).



The NP in (13) includes the adverbial *in a cafe*. The semantics of the modified VP is built as follows. Given the semantics of the indefinite article, which we will come to later, *a cafe* translates as (14):

---

distinction between functions and arguments.

[3] We could, if necessary, bring the semantics more in to line with the syntax. By generalising our definition of a category in (3), we could assign a complex category like S\NP a semantic variable distinct from those attached to its component atomic categories. This way the preposition would translate as:

(12)′ in := $((S\backslash NP)_{e_1}\backslash(S\backslash NP)_{e_1})/NP_{e_2} : [in(e_1,e_2)]$

In (12)′, the variable e_1 is used to represent the semantic constituent corresponding to S\NP (other variables have been omitted to make the category more readable). However, the limited types of semantic expression investigated in this thesis do not warrant such an accurate translation of modifiers; all that is required is a rough method by which sentences can pick out one known fact in accordance with the modifiers expressed within.

(14) a-cafe

NP_{e_1} : [cafe(e_1)]

The preposition *in*, as defined in (12), can combine with (14) by the forward application

((S\NP)\(S\NP))/NP NP ==> (S\NP)\(S\NP)

to produce (15), re-naming variables as necessary.

(15) in-a-cafe

$(S_{e_1}\backslash NP_{e_2})\backslash(S_{e_1}\backslash NP_{e_2})$: [in(e_1,e_3),cafe(e_3)]

Crossed backward composition (5c) is then invoked as follows,

(S\NP)/NP (S\NP)\(S\NP) ==> (S\NP)/NP,

composing the principal, right-hand functor in (15) with the subsidiary, left-hand functor in (11) to yield (16):

(16) ate-in-a-cafe

$(S_{e_1}\backslash NP_{e_2})/NP_{e_3}$: [eat(e_1,e_2,e_3),in(e_1,e_4),cafe(e_4)]

Thus, *ate in a cafe* translates into a logical form specifying that the eating event e_1 takes place in a cafe e_4. Now consider an appropriate context in which (13) might be evaluated:

(17) {eat(i1,jim,cake1), eat(i2,jim,cake2), ...,
in(i1,cafe1), in(i2,kitchen1), cafe(cafe1), kitchen(kitchen1)}

(17) includes two facts about Jim eating cake, indexed as *i1* and *i2*. Other predicates in the context can apply to these indices, to add further information about the individual events. So in (17), *in(i1,cafe1)* and *in(i2,kitchen1)* indicate that the two cake-eating events took place in different locations. Given this context, the constraints arising from (13), and in particular those in (16), will pick out *i1* as the relevant event, implying that *cake1* is the intended referent for the complete NP.

In general, indices like *i1* and *i2* will be treated as entities in the context, just like *jim* and *cake1*, but we will not bother to specify their properties, such as whether they denote events or states or whatever. All that matters is that there is some symbol distinguishing one fact from another, and so acting as a hook on which to hang any verbal modifier. As Davidson (1967) and Hobbs (1985) have shown, this approach allows an account of certain kinds of modification without complicating the logical translations by embedding one formula inside another.

Other lexical items, which we might very broadly class as *quantifiers*, follow a slightly different pattern to the above. For example, the definite determiner *the* translates into a different type of predicate to that characterising *man* and *eat*, and we will not discuss its

semantics until section 5.3. Other quantifiers simply map between variables, and introduce no explicit constraint of their own. This is true of the relative pronoun *which*, for example:

(18) a. which := $(N_{e_1}\backslash N_{e_1})/(S_{e_2}\backslash NP_{e_1})$: []

b. which := $(N_{e_1}\backslash N_{e_1})/(S_{e_2}/NP_{e_1})$: []

(18a) and (18b) capture the syntax and semantics of the subject relative *which* (as in *the rabbit which ate a carrot*), and the object relative *which* (as in *the carrot which the rabbit ate*) respectively. Thus (18a) is a function from a sentence looking for a subject NP into a modifying function over nouns, whereas (18b) is a function over sentences looking for objects. The distribution of semantic variables in both categories is the same, however; in each case the variable of the "missing" NP, e_1, is matched with that of the modified noun.

The first entry for *which* is used in the analysis of the fragment *rabbit which ate*. The type-raised version of the noun *rabbit* (19a) can compose with *which* (18a) to yield (19b). This in turn composes with *ate* (11) to produce (19c).

(19) a. rabbit

$N_{e_1}/(N_{e_1}\backslash N_{e_1})$: [rabbit(e_1)]

b. rabbit-which

$N_{e_1}/(S_{e_2}\backslash NP_{e_1})$: [rabbit(e_1)]

c. rabbit-which-ate

N_{e_1}/NP_{e_3} : [rabbit(e_1),eat(e_2,e_1,e_3)]

The indefinite article also bears no explicit constraint. Instead it relies on the implicit existential quantification of any variable mentioned in a constraint list. So the indefinite *a* appears as:

(20) a := NP_{e_1}/N_{e_1} : []

The definition in (20) states that the variable associated with the NP is the same as that associated with the noun, namely e_1. The indefinite may, for example, apply to a simple (i.e. non type-raised) noun such as *house* to form a complete NP:

(21) a-house

NP_{e_1} : [house(e_1)]

Subject type-raising, as revised below in (22), can turn an NP into a function over sentence predicates.

(22) Subject Type-Raising

$NP_v : T \uparrow S_w/(S_w\backslash NP_v) : T$

(Here, w and v range over semantic variables e_i.) Since determiners are functions into NPs, they bear an additional type-raised category derived from this rule. Thus, the indefinite article is also defined in the lexicon as

(23) a := $(S_{e_1}/(S_{e_1}\backslash NP_{e_2}))/N_{e_2} : []$

and, categorised as such, can apply to the noun *house* to yield

(24) a-house

$S_{e_1}/(S_{e_1}\backslash NP_{e_2}) : [house(e_2)]$

a type-raised subject NP.

We conclude this section with a worked example. Figure 5.1 shows how the noun phrase

(25) a man who visited a town near a river

is given a fully-incremental translation into the semantics presented above by the non-deterministic shift-reduce parser of Chapter 3. There is plenty of ambiguity in this string, but we will confine our attention to the most incremental reading provided by the grammar, which in this case allows the parser to make a reduction after every other step in processing. Note also that due to the option of type-raising noun phrases and nouns in the lexicon, the determiners and nouns in this example are in fact form-class ambiguous. In line with the discussion in Chapter 3, we will assume that the parser happens to shift the appropriate category for each lexical item in the string. We will therefore assume that all indefinite articles are functions into simple, non-type-raised NPs; and that the nouns *man* and *town* are type-raised over noun-modifying functions, with *river* in contrast treated as a simple noun. Similarly, we assume that the preposition *near* is given a noun-modifying reading, and that the relative pronoun *who* is read as introducing a subject relative clause. At each step in Figure 5.1, the contents of the stack are listed in an order which assumes that entries are pushed onto the stack at the *top*. Note also that the semantic variables in the stack are interpreted as local to each entry on the stack.

(a) Shift *a*

NP_{e_1}/N_{e_1} : []

(b) Shift *man*

$N_{e_1}/(N_{e_1}\backslash N_{e_1})$: [man(e_1)]
NP_{e_1}/N_{e_1} : []

(c) Reduce *a-man*

$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$: [man(e_1)]

(d) Shift *who*

$(N_{e_1}\backslash N_{e_1})/(S_{e_2}\backslash NP_{e_1})$: []
$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$: [man(e_1)]

(e) Reduce *a-man-who*

$NP_{e_1}/(S_{e_2}\backslash NP_{e_1})$: [man(e_1)]

(f) Shift *visited*

$(S_{e_1}\backslash NP_{e_2})/NP_{e_3}$: [visit(e_1,e_2,e_3)]
$NP_{e_1}/(S_{e_2}\backslash NP_{e_1})$: [man(e_1)]

(g) Reduce *a-man-who-visited*

NP_{e_1}/NP_{e_3} : [man(e_1),visit(e_2,e_1,e_3)]

(h) Shift *a*

NP_{e_1}/N_{e_1} : []
NP_{e_1}/NP_{e_3} : [man(e_1),visit(e_2,e_1,e_3)]

(i) Reduce *a-man-who-visited-a*

NP_{e_1}/N_{e_3} : [man(e_1),visit(e_2,e_1,e_3)]

(j) Shift *town*

$N_{e_1}/(N_{e_1}\backslash N_{e_1})$: [town(e_1)]
NP_{e_1}/N_{e_3} : [man(e_1),visit(e_2,e_1,e_3)]

(k) Reduce *a-man-who-visited-a-town*

$NP_{e_1}/(N_{e_3}\backslash N_{e_3})$: [man(e_1),visit(e_2,e_1,e_3),town(e_3)]

(l) Shift *near*

$(N_{e_1} \backslash N_{e_1})/NP_{e_2}$: [near(e_1,e_2)]
$NP_{e_1}/(N_{e_3} \backslash N_{e_3})$: [man(e_1),visit(e_2,e_1,e_3),town(e_3)]

(m) Reduce *a-man-who-visited-a-town-near*

NP_{e_1}/NP_{e_4} : [man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]

(n) Shift *a*

NP_{e_1}/N_{e_1} : []
NP_{e_1}/NP_{e_4} : [man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]

(o) Reduce *a-man-who-visited-a-town-near-a*

NP_{e_1}/N_{e_4} : [man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]

(p) Shift *river*

N_{e_1} : [river(e_1)]
NP_{e_1}/N_{e_4} : [man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]

(q) Reduce *a-man-who-visited-a-town-near-a-river*

NP_{e_1} : [man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4),river(e_4)]

Figure 5.1. Incremental translation of *a man who visited a town near a river*

5.2. Reference Evaluation

The previous section proposed a scheme of semantic translation suitable for incremental evaluation. We are now in a position to consider the evaluation process itself. The general approach will be to associate a semantic evaluation with every constituent found by the parser, lexical or otherwise. Thus, whenever a word is shifted by the parser, its lexical constraint list will be evaluated into a constraint network and its consistency determined. Consequently, when the parser reduces two stack entries they will already be linked to semantic evaluations. Producing a semantic evaluation for the reduced constituent therefore consists of combining the networks of the two component constituents in line with the reduction, and propagating consistency within this composite network.

We will thus have a scheme of reference evaluation which might loosely be described as being "compositional", in that each constituent has an evaluation and the evaluation of

a whole is a simple combination of the evaluations of its parts. All that then has to be done to effect an *incremental* evaluation strategy is to ensure that the overseeing syntactic processor reads from left to right and makes reductions as soon as possible.

The following two sections present some of the algorithmic detail necessary to connect the semantic translation process with the network consistency procedure of Chapter 4. There are a number of ways of implementing the connection between semantics and reference, and the proposal below is designed to emphasise *what* referential work may be done at each Shift and Reduce point, rather than *how* it should be done. Once the mechanisms are in place, they are illustrated by the incremental evaluation of an indefinite noun phrase taken to refer anaphorically to the context.

5.2.1. Evaluating a Shift

The constraints from which a network is derived originate in the lexicon, in constraint lists associated with particular words. So, whenever a word is read, and one of its lexical entries is shifted onto the stack, its constraint list is evaluated to yield a network of constraints. Assuming the network is consistent, it is then pushed onto the stack along with the other fields of the word's description. We therefore revise the Shift action of Chapter 3 as in Figure 5.2 below.

The new Shift operation evaluates a lexical constraint list by calling the procedure Create Network at step (3). Create Network is essentially the algorithm Mackworth from Chapter 4; it takes a constraint list and, for each constraint in this list, computes a set of arcs, and propagates consistency. If the procedure determines an inconsistency, it fails; otherwise it returns the consistent network — consisting of the set of arcs and value

---

Shift. Do steps (1)-(5):

(1) Remove the next lexical item from the input buffer.

(2) Find a lexical entry for the item with category C and translation T.

(3) Create Network from T, to yield consistent network N.

(4) Push the entry <C,T,N> onto the stack.

(5) Call Parse.

Figure 5.2. Evaluation incorporated into the Shift operation.

---

domains — for use in later evaluations. For the interested reader, this evaluation algorithm is presented in Figure 5.3. (Note that arcs arising from unary constraints are not saved, since, in principle, once they have been applied to a given value domain they cannot cause any further revisions to that domain. Also note that in Propagate, which propagates operations of domain refinement, it is assumed that whenever the procedure encounters a new variable, it invokes a fresh domain initially consisting of all contextual entities.)

To appreciate the actions of the above processes, suppose a phrase is being parsed against the context in (26).

(26) {pyramid(p1), pyramid(p2), pyramid(p3), pyramid(p4),
box(b1), box(b2), box(b3), box(b4),
blue(p1), blue(p2), blue(p4), blue(b1),
in(p1,b1), in(p2,b2), in(p3,b3), in(b4,b1)}

If, at some stage, the processor reads the noun-modifying reading of the preposition *in*, defined earlier in (9), the effect of the semantically evaluating Shift operation will be to push the following entry onto the stack:

Create Network from T. Do steps (1)-(3):

(1) Initialise set of arcs Arcs to $\varnothing$.

(2) For each constraint C in T, do steps (2.1)-(2.4):

  (2.1) Initialise arc queue Q to $\varnothing$.

  (2.2) For each argument e_i of C, add the pair $<i,C>$ to Q.

  (2.3) If C is not a unary constraint, add Q to Arcs.

  (2.4) Propagate from Q.

(3) Return the set Arcs and the currently applicable set of value domains.

Propagate from Q. While Q is not empty, do steps (1)-(3):

(1) Remove the next arc $<i,C>$ from Q.

(2) Reset $D_i \leftarrow$ REFINE(i,C).

(3) If D_i was reduced in size by step (2), then do step (3.1):

  (3.1) If D_i is now empty then FAIL,
  else reset $Q \leftarrow Q \cup \{<j,C'> \mid <j,C'> \in \text{Arcs}, C' \text{ constrains } i, i \neq j, C' \neq C\}$

Figure 5.3. Evaluation of a constraint-list translation T.

---

(27) $(N_{e_1}\backslash N_{e_1})/NP_{e_2}$
[in(e_1,e_2)]
([<1,in(e_1,e_2)>, <2,in(e_1,e_2)>],
[D_1 : {p1,p2,p3,b4}, D_2 : {b1,b2,b3}])

In subsequent examples we will usually omit the arc set of the constituent, and simply summarise the network by its domains. So, unless we are specifically interested in the arc information, stack entries will be written according to the more readable form in (27)':

(27)' $(N_{e_1}\backslash N_{e_1})/NP_{e_2}$
[in(e_1,e_2)]
[D_1 : {p1,p2,p3,b4}, D_2 : {b1,b2,b3}]

Here, then, we can regard the constraint list [in(e_1,e_2)] as a shorthand for the arcs in question.

5.2.2. Evaluating a Reduction

Whenever the stack reduces, the top two categories and their constraint list translations are combined in the manner specified by the grammar rules of section 5.1. In addition, a semantic evaluation can be associated with these reduced structures by combining the top two constraint network evaluations in accordance with the grammatical rewrite. For instance, consider the processing of the fragment *blue pyramid in*, where the relevant context is (26) and two shifts, on *blue* and a type-raised *pyramid*, have produced the top two stack entries in (28). (As always the variables and their domains are local to each entry, but in this example their names have been distinguished to clarify the exposition.)

(28) $N_{e_2}/(N_{e_2}\backslash N_{e_2})$
[pyramid(e_2)]
[D_2 : {p1,p2,p3,p4}]

N_{e_1}/N_{e_1}
[blue(e_1)]
[D_1 : {p1,p2,p4,b1}]

Category reduction of (28) invokes forward composition, unifying N_{e_1} and N_{e_2}. Since e_1 and e_2 are equated at the category level, it seems clear that their respective value domains should be *intersected* at the network level. So the following reduced stack entry should be derived from (28):

(29) $N_{e_1}/(N_{e_1}\backslash N_{e_1})$
[blue(e_1),pyramid(e_1)]
[D_1 : {p1,p2,p4}]

This serves to illustrate the essence of network combination. If any two variables are unified at the semantic level, then their domains are intersected at the referential level. In (28), if D_1 is consistent with *blue*(e_1) and D_2 is consistent with *pyramid*(e_2), then with $e_1 = e_2$ the intersection of D_1 and D_2 must be consistent with the conjunction of the constraints.

In the general case, a combinatory rule may unify several pairs of semantic variables in a single rewrite operation. And whenever this occurs, the corresponding pairs of value domains must be intersected. So the domains and arcs pertaining to the combined network are just those pertaining to the two component networks, collected together; except, when two variables e_{i_1} and e_{i_2} have been unified by the category reduction, the intersection of D_{i_1} and D_{i_2} replaces D_{i_1} and D_{i_2} in the combined set of domains. Furthermore, if domains are intersected in this manner, neighbouring nodes must be checked for consistency. Intersection preserves consistency, but may, inadvertently, make neighbouring nodes inconsistent. Propagation therefore starts with a queue Q of those arcs directly related to the altered value domains; i.e. the queue is initialised as

$$Q \leftarrow \{<j,C> \mid <j,C> \in \text{Arcs, C constrains } i,\ i \neq j\}$$

where i is any node which has been unified with some other.

The algorithms in Figures 5.4 and 5.5 summarise the way in which extensions are evaluated at points of stack reduction; let us now see how they work in relation to the example in (28) and (29). Suppose that the parser proceeds from the state in (29) by reading another word, the preposition *in*. Shifting its definition in (9) would produce the stack configuration in (30).

Reduce. Do steps (1)-(6):

(1) Pop the top entry <C2,T2,N2> from the stack.
(2) Pop the next entry <C1,T1,N1> from the stack.
(3) Find a rule s.t. C1:T1 C2:T2 ==> C:T.
(4) Merge Networks N1 and N2 to yield a consistent network N.
(5) Push the entry <C,T,N> onto the stack.
(6) Call Parse.

Figure 5.4. Evaluation incorporated into the Reduce operation.

Merge Networks N1 and N2. Do steps (1)-(4):

(1) Concatenate arcs of N1 and arcs of N2 to form the collective set Arcs.

(2) For any nodes i now occurring in both networks N1 and N2, where D_{i_1} is the domain of e_i in N1 and D_{i_2} is the domain of e_i in N2, form a single domain D_i consisting of their intersection; i.e. set $D_i \leftarrow D_{i_1} \cap D_{i_2}$. If $D_i = \varnothing$ then FAIL.

(3) If any domains have been revised by step (2), then do steps (3.1)-(3.3):

  (3.1) Initialise Q to $\varnothing$.

  (3.2) For each revised domain D_i, do step (3.2.1):

    (3.2.1) Reset $Q \leftarrow Q \cup \{<j,C> \mid <j,C> \in$ Arcs, C constrains i, $i \neq j\}$

  (3.3) Propagate from Q.

(4) Return the set Arcs and the currently applicable set of value domains.

Figure 5.5. Merging networks.

(30) $(N_{e_2} \backslash N_{e_2})/NP_{e_3}$
[in(e_2,e_3)]
([<2,in(e_2,e_3)>, <3,in(e_2,e_3)>],
[D_2 : {p1,p2,p3,b4}, D_3 : {b1,b2,b3}])

$N_{e_1}/(N_{e_1} \backslash N_{e_1})$
[blue(e_1),pyramid(e_1)]
([],
[D_1 : {p1,p2,p4}])

(The arcs associated with each constituent are included in (30) to make the example easier to understand.) Applying Reduce to the stack in (30) rewrites the categories and constraint list translations by forward composition. This rewrite unifies the variables e_1 and e_2 and, assuming that the variables unify to e_1, produces the category N_{e_1}/NP_{e_3}. So Merge Networks must now intersect D_1 and D_2. After the intersection of step (2), but before the propagation of step (3), this yields the composite network of (31):

(31) ([<1,in(e_1,e_3)>, <3,in(e_1,e_3)>], [D_1 : {p1,p2}, D_3 : {b1,b2,b3}])

At this point, D_1 is consistent because it arose from the intersection of two consistent domains, D_1 and D_2. But the directly related domain D_3 must now be refined so that each of its values contains something in the new D_1. Hence the initial queue is {<3,in(e_1,e_3)>}. Domain refinement resets D_3 to {b1,b2}, emptying the queue and yielding a consistent network. The parser finally pushes this onto the stack with the category and translation, to give the entry in (32), representing the string *blue pyramid in*.

(32) N_{e_1}/NP_{e_3}
[blue(e_1),pyramid(e_1),in(e_1,e_3)]
([<1,in(e_1,e_3)>, <3,in(e_1,e_3)>],
[D_1 : {p1,p2}, D_3 : {b1,b2}])

There are two points to note about this formalisation of the evaluation process. First, it is perhaps not as elegant as it might be. The unificational operations on semantic variables at the category level — from the rewrite of step (3) in Figure 5.4 — are allowed to side-effect the names of nodes in the constraint network. The algorithm Merge Networks then looks for these side-effects (at step (2), Figure 5.5) and intersects those pairs of domains which are now associated with the same variable. A better approach would be to explicitly parameterise the combinatory rules, so that, rather than destructively unifying variables, they return a declarative constraint indicating the semantic effect of the combination. For example, composing *blue* and *pyramid* in (28) would yield the declarative statement "$e_1 = e_2$". This could then be passed on to Merge Networks as an explicit guideline for merging the networks, so avoiding the side-effecting properties of the present algorithm. (A variant of this approach was adopted in Haddock (1987) — see Appendix B.) However, such equality conditions would complicate the semantic translation process, and so this procedural inelegancy is accommodated here in order to retain simplicity of exposition elsewhere in the thesis.

Second, we can see from the examples above that the constraint-list translation and the derived arc set are really one and the same thing. In other words, we could discard each lexical constraint list once it has been turned into a constraint network, at a Shift point, since subsequent evaluations depend only on the derived arcs and domains. Alternatively, we could keep the semantic translation and instead discard the arcs at each stage, reproducing them from the translation when necessary. But it seems harmless enough to keep building up both representations, even if the only function of this redundancy is to make our examples easier to follow.

5.2.3. An Example of Incremental Evaluation

In summary, the processor evaluates the semantics of any new constituent on its stack. Lexical representations are evaluated in terms of their semantic translations, and composite representations are evaluated by combining the evaluations of their parts. Of course, the degree of *left-to-right* incremental evaluation in the system is determined completely by the parsing strategy employed. If the strategy is conceived so as to derive right-

branching syntactic analyses (i.e. trying to shift before trying to reduce), then semantic evaluation will accordingly proceed from right to left. But we are more interested in left-to-right parsing, and so the following example of incremental processing again assumes that reductions are made as early as possible. Figure 5.6 details the state of evaluation at each Shift and Reduce point in a parse of *a man who visited a town near a river*, the noun phrase previously used to exemplify incremental translation in Figure 5.1. The relevant context is shown in (33).

(33) { man(m1),
man(m2),
man(m3),
man(m4),
man(m5),
man(m6),
woman(w1),
town(t1),
town(t2),
town(t3),
town(t4),
town(t5),
village(v1),
river(r1),
river(r2),
lake(l1),
visit(i1,m1,t1),
visit(i2,m1,t3),
visit(i3,m2,t1),
visit(i4,m2,t2),
visit(i5,m3,v1),
visit(i6,m5,t3),
visit(i7,m6,t4),
visit(i8,w1,t5),
near(t1,r1),
near(t2,r1),
near(t3,l1),
near(t5,r2),
near(v1,r2) }

(a) Shift *a*

NP_{e_1}/N_{e_1}
[]
[]

(b) Shift *man*

$N_{e_1}/(N_{e_1}\backslash N_{e_1})$
[man(e_1)]
[D_1 : {m1,m2,m3,m4,m5,m6}]
NP_{e_1}/N_{e_1}
[]
[]

(c) Reduce *a-man*

$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[man(e_1)]
[D_1 : {m1,m2,m3,m4,m5,m6}]

(d) Shift *who*

$(N_{e_1}\backslash N_{e_1})/(S_{e_2}\backslash NP_{e_1})$
[]
[]
$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[man(e_1)]
[D_1 : {m1,m2,m3,m4,m5,m6}]

(e) Reduce *a-man-who*

$NP_{e_1}/(S_{e_2}\backslash NP_{e_1})$
[man(e_1)]
[D_1 : {m1,m2,m3,m4,m5,m6}]

(f) Shift *visited*

$(S_{e_1}\backslash NP_{e_2})/NP_{e_3}$
[visit(e_1,e_2,e_3)]
[D_1 : {i1,i2,i3,i4,i5,i6,i7,i8}, D_2 : {m1,m2,m3,m5,m6,w1}, D_3 : {t1,t2,t3,t4,t5,v1}]
$NP_{e_1}/(S_{e_2}\backslash NP_{e_1})$
[man(e_1)]
[D_1 : {m1,m2,m3,m4,m5,m6}]

(g) Reduce *a-man-who-visited*

NP_{e_1}/NP_{e_3}
[man(e_1),visit(e_2,e_1,e_3)]
[D_1 : {m1,m2,m3,m5,m6}, D_2 : {i1,i2,i3,i4,i5,i6,i7}, D_3 : {t1,t2,t3,t4,v1}]

(h) Shift *a*

NP_{e_1}/N_{e_1}
[]
[]
NP_{e_1}/NP_{e_3}
[man(e_1),visit(e_2,e_1,e_3)]
[D_1 : {m1,m2,m3,m5,m6}, D_2 : {i1,i2,i3,i4,i5,i6,i7}, D_3 : {t1,t2,t3,t4,v1}]

(i) Reduce *a-man-who-visited-a*

NP_{e_1}/N_{e_3}
[man(e_1),visit(e_2,e_1,e_3)]
[D_1 : {m1,m2,m3,m5,m6}, D_2 : {i1,i2,i3,i4,i5,i6,i7}, D_3 : {t1,t2,t3,t4,v1}]

(j) Shift *town*

$N_{e_1}/(N_{e_1}\backslash N_{e_1})$
[town(e_1)]
[D_1 : {t1,t2,t3,t4,t5}]
NP_{e_1}/N_{e_3}
[man(e_1),visit(e_2,e_1,e_3)]
[D_1 : {m1,m2,m3,m5,m6}, D_2 : {i1,i2,i3,i4,i5,i6,i7}, D_3 : {t1,t2,t3,t4,v1}]

(k) Reduce *a-man-who-visited-a-town*

$NP_{e_1}/(N_{e_3}\backslash N_{e_3})$
[man(e_1),visit(e_2,e_1,e_3),town(e_3)]
[D_1 : {m1,m2,m5,m6}, D_2 : {i1,i2,i3,i4,i6,i7}, D_3 : {t1,t2,t3,t4}]

(l) Shift *near*

$(N_{e_1}\backslash N_{e_1})/NP_{e_2}$
[near(e_1,e_2)]
[D_1 : {t1,t2,t3,t5,v1}, D_2 : {r1,r2,l1}]
$NP_{e_1}/(N_{e_3}\backslash N_{e_3})$
[man(e_1),visit(e_2,e_1,e_3),town(e_3)]
[D_1 : {m1,m2,m5,m6}, D_2 : {i1,i2,i3,i4,i6,i7}, D_3 : {t1,t2,t3,t4}]

(m) Reduce *a-man-who-visited-a-town-near*

NP_{e_1}/NP_{e_4}
[man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]
[D_1 : {m1,m2,m5}, D_2 : {i1,i2,i3,i4,i6}, D_3 : {t1,t2,t3}, D_4 : {r1,l1}]

(n) Shift *a*

NP_{e_1}/N_{e_1}
[]
[]
NP_{e_1}/NP_{e_4}
[man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]
[D_1 : {m1,m2,m5}, D_2 : {i1,i2,i3,i4,i6}, D_3 : {t1,t2,t3}, D_4 : {r1,l1}]

(o) Reduce *a-man-who-visited-a-town-near-a*

NP_{e_1}/N_{e_4}
[man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]
[D_1 : {m1,m2,m5}, D_2 : {i1,i2,i3,i4,i6}, D_3 : {t1,t2,t3}, D_4 : {r1,l1}]

(p) Shift *river*

N_{e_1}
[river(e_1)]
[D_1 : {r1,r2}]
NP_{e_1}/N_{e_4}
[man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4)]
[D_1 : {m1,m2,m5}, D_2 : {i1,i2,i3,i4,i6}, D_3 : {t1,t2,t3}, D_4 : {r1,l1}]

(q) Reduce *a-man-who-visited-a-town-near-a-river*

NP_{e_1}
[man(e_1),visit(e_2,e_1,e_3),town(e_3),near(e_3,e_4),river(e_4)]
[D_1 : {m1,m2}, D_2 : {i1,i3,i4}, D_3 : {t1,t2}, D_4 : {r1}]

Figure 5.6. Incremental evaluation of *a man who visited a town near a river*

5.3. Closure Constraints

We have now connected the syntactic and referential elements of the model and so have provided a basis for incremental semantic evaluation. If we are to apply the system to the problematic rabbit phrases and use it for local syntactic ambiguity resolution, we must, in addition, consider the semantic role of the definite article. This section presents the

case for the definite determiner introducing an additional kind of semantic constraint, called a *closure constraint*, and shows how it can be incorporated into the model as it stands.

5.3.1. Semantics of the Definite Article

The category of the definite article *the*, NP_{e_1}/N_{e_1}, conforms to the pattern set by the indefinite in (20). Unlike the indefinite, however, the definite determiner yields an explicit constraint, *unique(e_1)*:

(34) the := NP_{e_1}/N_{e_1} : [* unique(e_1)]

This constraint will, in due course, be a termed a *closure constraint*. (For the time being, ignore the asterisk.) The predicate *unique* is to be interpreted with respect to some particular constraint network according to the following definition in (35).

(35) unique(e_i) iff $|D_i| = 1$

So *unique(e_i)* holds when the value domain of e_i, D_i, contains just one element.

5.3.2. Differences in Type and Timing

The uniqueness constraint above is different in kind from the constraints arising from "non-quantifiers" like *rabbit*, *in* and *eats*. Whereas a constraint like *rabbit(e_i)* applies a predicate *rabbit* to each of the particular entities in the domain of e_i, *unique* is a predicate over the domain as a whole. *Rabbit* is a predicate over entities; *unique* is a predicate over sets of entities.

This has implications for the evaluation of (35). The present system maintains domains consisting of values which correspond to individual entities in a context. Any constraint administered by the consistency algorithm, and so input to the REFINE procedure, is therefore expected to specify a predicate over individual entities. But *unique* is not a predicate over entities; it is a predicate over sets of entities. With this in mind, we should not regard *unique(e_i)* as an ordinary constraint, but as a "meta-constraint". Rather than filtering out individual entities from a given value domain, it makes a check on *the degree to which the domain is constrained.*

So, the definite article does not provide referentially constraining information in the sense of *rabbit* and *in*. This perspective underlies many other accounts of definite reference; for instance, Winograd's treatment of definite full NPs. The point is a subtle one, and is best illustrated by an example which supports the present position. Consider the

context pictured in Figure 5.7. Two blocks, *block1* and *block2*, sit on a tray *tray1*, and a second tray *tray2* supports another block, *block3*. In the light of Figure 5.7, now consider the referring expressions in (36).

(36) a. # the block on the tray
b. # the block on a tray

Neither of the phrases in (36) seems to be a felicitous means of picking out the third block *block3*. And this infelicity is predicted by a reference scheme which does not regard definiteness as constraining information; for the constraints [block(e_1), on(e_1,e_2), tray(e_2)] determine three possible blocks, not one. However, an approach which somehow treated definiteness as a constraint on the candidate contextual entities might well determine, undesirably, a singleton set of blocks: *there is* a unique block such that it is the *only* block sitting on a specific tray (i.e. *block3*).

We have so far found two essential differences between the uniqueness check and the constraints arising from non-quantifiers. First, *unique* predicates sets whereas *rabbit* predicates individuals. Second, because the consistency algorithm deals with constraints over individual entities, the singleton check is *not* seen as a constraint to be satisfied by the algorithm but instead as a kind of meta-level check on the results of domain refinement.

Another, logically independent, aspect of (35) distinguishes it from the rest. In a system which aims to evaluate reference early, it seems natural to evaluate nouns and their modifiers as they are read. It makes little sense to do the same with a definite article. Presumably the whole point of the words which make up the rest of a definite noun phrase is to help identify a unique referent. We should not necessarily expect the constraints incrementally arising from a definite NP to yield a unique referent until (at least) the NP is syntactically complete; not if we are to make theoretical predictions about its felicity as a

Figure 5.7.

whole.

The proposal advanced here, and explored further in Chapter 7, will be that *a definite NP should refer uniquely by the time it is syntactically closed.* So, in a definite NP such as *the red block*, the uniqueness constraint checks for the singleton set *after* the predicates *red* and *block* have incrementally refined the domain of candidates for the NP. (If the check fails, the parser gives up on this reading of the string.) It is for this reason that the constraint in (34) is called a *closure constraint.* And to make the processor aware of this difference, we use a "*" prefix in the lexical translation to identify *unique(e_i)* as a closure constraint. Accordingly, we will now distinguish ordinary constraints like *rabbit(e_i)*, which are an integral part of a constraint network, as *network constraints.*[4]

5.3.3. Timing the Evaluation of Closure Constraints: Function Application and Noun Phrase Closure

So closure constraints differ in both type and timing to ordinary network constraints. That closure constraints are timed to evaluate at a different point to ordinary constraints poses a technical problem which threatens the simplicity of the model as it stands. It is easy to characterise the timing of a constraint which must be evaluated as soon as possible: as section 5.2 demonstrated, evaluation coincides with shifting the corresponding word. But the application of a closure constraint must be delayed until the corresponding noun phrase is closed. And as the parser does not build any syntactic structure, how is it to know when a particular NP is syntactically complete?

Luckily, we do not have to resort to building phrase-structure trees to answer this question. We can, instead, exploit the difference between the two combinatory rules which combine categories. Function *composition* can never close a noun phrase. NP closure must always coincide with some or other function *application* made in the course of an analysis.

Consider the categories of nouns. As a simple N, a noun can combine with a determiner NP/N to its left to form a closed, NP constituent — by function application.

[4] Although we only consider singular NPs here, it seems likely that this treatment of the definite article will also be appropriate for plural NPs like *the blocks*. In this framework, plural expressions might evaluate into domains whose values are names of sets of entities rather than simply entities. Thus *blocks(e_1)* might produce a domain D_1 = {set3}, where *set3* identifies some set of blocks (perhaps determined by considering the context, the enclosing discourse, speaker/hearer intentions, and so on). With this approach, the meta-level check for uniqueness defined in (35) would apply to this domain just as it would to any other, and achieve the desired effect of checking that there is a unique set of blocks; it doesn't need to be sensitive to the fact that in the plural case the values of the domain are sets of actual entities, whereas in the singular case they are just entities.

Alternatively, type-raising N to a function over modifying categories N/(N\N) forces the NP to stay open, and the noun can combine with an NP/N on the left only by function composition.

The rule that noun phrase closure always coincides with function application appears to be a general one, at least when parsing from left to right. Thus, a left-to-right incremental analysis of *the block in the box*, with *block* type-raised, involves three compositions followed by an application. The final application, to *box*, will in this case close two NPs, *the box* and *the block in the box*. To take another example, consider the following derivation of *the gate which swings*:

```
(37)  the      gate          which          swings
      ----     -------     -----------      -------
      NP/N   N/(N\N)     (N\N)/(S\NP)        S\NP
      ------------->c
         NP/(N\N)
         -------------------------->c
                  NP/(S\NP)
                  --------------------------->a
                             NP
```

Here, *the gate which swings* is closed by the application of NP/(S\NP) (*the gate which*) to S\NP (*swings*).

So the evaluation of closure constraints can be triggered by function application. The processor simply has to maintain a list of unevaluated closure constraints, a list formed by the composition of the associated constituents, and wait until a subsequent function application authorises their enforcement. Hence a constituent is now characterised by a *four-part* representation: its category, its list of closure constraints (the *closure list*), its ordinary, network-oriented translation, and its constraint network.

For example, the definite determiner *the* and the adjective *red* appear, uncombined, on the stack as

(38) N_{e_1}/N_{e_1}
[]
[red(e_1)]
[D_1 : {b1,b2}]

NP_{e_1}/N_{e_1}
[unique(e_1)]
[]
[]

if *b1* and *b2* are the only red things in the context. Note that *red* has an empty closure list

because it does not give rise to any closure constraints. Applying function composition to the stack in (38) allows category cancellation, concatenation of the translations, and network combination in the usual way. In addition, the two closure lists are concatenated. Thus the composition of *the* and *red* yields the entry in (39):

(39) NP_{e_1}/N_{e_1}
[unique(e_1)]
[red(e_1)]
[D_1 : {b1,b2}]

The closure list for *the block in the* is built up in a similar way:

(40) NP_{e_1}/N_{e_2}
[unique(e_1),unique(e_2)]
[block(e_1),in(e_1,e_2)]
[D_1 : {...}, D_2 : {...}]

Hence function composition concatenates closure lists as well as the ordinary constraint lists. In fact, the rule of function application should do the same: it should concatenate the closure list associated with the function category with the closure list associated with the argument category.[5] And, by the argument presented above, each constraint in a closure list is evaluated just after some or other function application involving that closure list. But how can the parser tell *which* constraints in the closure list (if any) are to be enforced at the time of a particular application? One simple assumption, explicated below, enables the parser to discharge the entire, concatenated list at the *first* function application.

Returning to (40), the application of *the block in the* to *box*, characterised in (42)

(42) N_{e_1}
[]
[box(e_1)]
[D_1 : {...}]

produces an NP_{e_1} with the interim closure list

[unique(e_1),unique(e_2)]

At this stage, because function application was used, an attempt is made to satisfy each of the constraints in the closure list. The closure constraint *unique(e_1)* is satisfied if the domain of e_1 in the associated network (not specified) has cardinality 1. The same

[5] Typically, however, the *argument* in a function application has a null closure list. However, potentially it is useful to consider the possibility of closure constraints arising from an argument because of certain atomic lexical categories which bear closure constraints. For instance, if proper names are expected to refer uniquely they might be defined as:

(41) john := NP_{e_1} : [name(e_1,"john"), * unique(e_1)]

conditions hold for *unique(e_2)*.

Two minor points should be noted here. First, the process of successfully satisfying the definiteness check does not bind a variable to the single value in its domain. There is no need to do this and, furthermore, it would complicate the consistency algorithm. Second, a closure constraint is not kept once it has been evaluated. There is no need to re-check for the singleton set later on because the set can only be reduced; and any further reduction of a singleton set would necessarily violate the existential quantification implicit in the consistency algorithm.

We conclude with the relatively simple matter of re-defining the rules and algorithms in the model to incorporate closure constraints. First, the combinatory rules should be trivially extended so that they concatenate closure lists as well as ordinary constraint lists. So function application and composition should be revised to (43) and (44), respectively,

(43) Forward and Backward Application

a. X/Y : U1 : T1 Y : U2 : T2 ==> X : (U1 + U2) : (T1 + T2)
b. Y : U1 : T1 X\Y : U2 : T2 ==> X : (U1 + U2) : (T1 + T2)

(44) Forward and Crossed Backward Composition

a. X/Y : U1 : T1 Y/Z : U2 : T2 ==> X/Z : (U1 + U2) : (T1 + T2)
b. Y/Z : U1 : T1 X\Y : U2 : T2 ==> X/Z : (U1 + U2) : (T1 + T2)

where U1 represents the left-hand closure list and U2 represents the right-hand closure list. (We will come to the similarly revised rule of generalised forward composition in a moment.)

The Shift procedure must therefore form two distinct lists from a lexical translation: one containing the closure constraints U and one containing the ordinary, network constraints T. A four-part entry is then shifted onto the stack: the category C, the closure list U, the network constraint list T, and the evaluation of T, the network N. The revised algorithm appears in Figure 5.8.

Finally, the Reduce algorithm is revised as in Figure 5.9. After network evaluation, in step (5) the procedure discharges the closure list if the reduction was sanctioned by a rule of function application; i.e. it enforces each of the closure constraints in the list U, and then resets U to the empty list. Should the requirements of any of these closure constraints fail to be met, the operation fails.

Shift. Do steps (1)-(5):

(1) Remove the next lexical item from the input buffer.

(2) Find a lexical entry for the item with category C, closure constraints U, and network constraints T.

(3) Create Network from T, to yield consistent network N.

(4) Push the entry <C,U,T,N> onto the stack.

(5) Call Parse.

Figure 5.8. Closure constraints incorporated into the Shift operation.

Reduce. Do steps (1)-(7):

(1) Pop the top entry <C2,U2,T2,N2> from the stack.

(2) Pop the next entry <C1,U1,T1,N1> from the stack.

(3) Find a rule s.t. C1:U1:T1 C2:U2:T2 ==> C:U:T.

(4) Merge Networks N1 and N2 to yield a consistent network N.

(5) If the rule used in step (3) was function application, then discharge U.

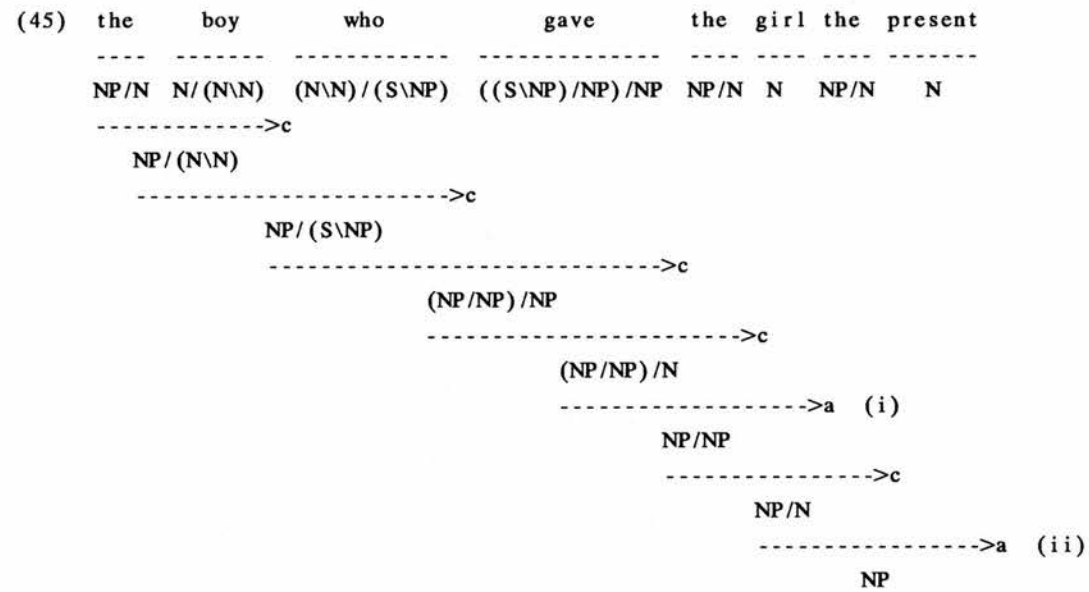
(6) Push the entry <C,U,T,N> onto the stack.

(7) Call Parse.

Figure 5.9. Closure constraints incorporated into the Reduce operation.

5.3.4. Generalising the Account

We now turn to the assumption, mentioned above, which must accompany this approach to the timing of closure constraint evaluation. The assumption is that generalised function composition rules, such as the rule in (5b) above, are not included in the rule set. To see why these rules can create problems, consider the following NP analysis:



This analysis uses the rule of generalised forward composition to combine *the boy who*, an NP/(S\NP), with the ditransitive verb *gave*, an ((S\NP)/NP)/NP, to form (NP/NP)/NP. Consequently, by the forward composition rule of (44a), *the boy who gave the* is analysed as (NP/NP)/N. The problem is that the subsequent *application* (marked as application (i) in (45)) of (NP/NP)/N to N closes the NP *the girl*, but does not close the NP *the boy who* ..., since the latter, complex NP still requires a direct object. This flaws the simple scheme presented above, which, after the application to *girl*, would discharge both the closure constraint related to *the girl*, *and* that related to *the boy*. Instead, it should enforce only the closure constraint related to *the girl* at point (i); the other closure constraints should not be discharged until the final application at (ii).

Generalised composition lies at the root of this problem. Consider the category associated with the NP fragment *the boy who*: as an NP/(S\NP), it needs a constituent of category S\NP to form a complete, closed NP. Since this is the only argument required to complete the NP, the closure constraint associated with the domain of boys should be discharged after one function application. The basic, first-order rule of forward composition in (44a) preserves this state of affairs; compose NP/(S\NP) with some (S\NP)/Z to its right (e.g. a transitive verb like *ate*) and the larger fragment NP/Z will still require only one argument (Z) before this closure constraint can be discharged. But the generalised rule of (5b), being an instance of second-order composition, does not keep things as they are. Whereas $NP_i/(S\backslash NP)$ requires one argument to form a complete NP_i, the category which results from generalised composition with *gave*, $(NP_i/NP_j)/NP_k$, requires two arguments to form a complete NP_i. So whereas in the first case the closure constraint

associated with NP_i can be discharged after one application, in the second case it must wait until two applications have been made.

Let us assume that the model does in fact include a rule of generalised composition, and that (5b) is revised in line with the other rules, as follows:

(46) Generalised Forward Composition

X/Y : U1 : T1 (Y/W)/Z : U2 : T2 ==> (X/W)/Z : (U1 + U2) : (T1 + T2)

There are then a number of ways of enriching the representations already in place to accommodate the effects of generalised composition, and one of these is to index the constraints in the closure list so that each of them is discharged at the appropriate moment in processing. We can index each closure constraint simply by associating it with an integer representing the number of syntactic arguments which must be found (or, equivalently, the number of function applications which must be made) before it should be enforced. So the definite article now translates as (34)'

(34)' the := NP_{e_1}/N_{e_1} : [* unique$(e_1)^1$]

where the superscripted "1" means that the uniqueness check should be made at the first function application. We will refer to such superscripted indices as *closure points*.

How do function application and composition combine the indexed closure lists of their operands? They will concatenate the lists, just as before. But if function composition is used to link the lists, the closure points of the constraints associated with the *principal* functor must be adjusted in line with the order of composition. In general, where the order of composition is n, the processor must increase the closure points of each of the closure constraints associated with the principal functor by n - 1. Since the canonical rules of composition in (44) are first-order rules, they set $n = 1$ and so leave the closure points unchanged. But as the rule of (46) is an instance of second-order composition, here $n = 2$ and each of the closure points in the closure list of the principal functor, U1, gets incremented by 1. Finally, whenever function application is invoked, all closure constraints with a closure point of 1 are discharged; and the closure points of the remaining closure constraints are all decremented by 1.

To see how this scheme works in practice, consider again the example in (45). Under the new regime, the initial determiner will appear on the stack as (47a), evaluation details omitted.

(47) a. Shift *the*

NP_{e_1}/N_{e_1}
[unique$(e_1)^1$]
[]

The subsequent noun *boy* introduces no closure constraints of its own, and since first-order composition does not affect closure points, *the* and *boy* reduce as follows:

(47) b. Reduce *the-boy*

$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[unique$(e_1)^1$]
[boy(e_1)]

And for simplicity ignoring the semantic variable associated with the S, *the boy who* is analysed as in (47c).

(47) c. Reduce *the-boy-who*

$NP_{e_1}/(S\backslash NP_{e_1})$
[unique$(e_1)^1$]
[boy(e_1)]

The verb *gave* is shifted to produce the stack configuration

(47) d. Shift *gave*

$((S\backslash NP_{e_1})/NP_{e_3})/NP_{e_2}$
[]
[gave(e_1,e_2,e_3)]
$NP_{e_1}/(S\backslash NP_{e_1})$
[unique$(e_1)^1$]
[boy(e_1)]

and its subsequent combination with *the boy who* by second-order composition yields:

(47) e. Reduce *the-boy-who-gave*

$(NP_{e_1}/NP_{e_3})/NP_{e_2}$
[unique$(e_1)^2$]
[boy(e_1),gave(e_1,e_2,e_3)]

Note that the use of second-order composition has incremented the closure point attached to *unique*(e_1). However, the closure point of the next incoming determiner remains at 1:

(47) f. Shift *the*

NP_{e_1}/N_{e_1}
[unique$(e_1)^1$]
[]
$(NP_{e_1}/NP_{e_3})/NP_{e_2}$
[unique$(e_1)^2$]
[boy(e_1),gave(e_1,e_2,e_3)]

g. Reduce *the-boy-who-gave-the*

$(NP_{e_1}/NP_{e_3})/N_{e_2}$
[unique$(e_1)^2$,unique$(e_2)^1$]
[boy(e_1),gave(e_1,e_2,e_3)]

The following function application to *girl* thus discharges *unique(e_2)*, but leaves *unique(e_1)* where it is, simply decrementing its closure point:

(47) h. Reduce *the-boy-who-gave-the-girl*

NP_{e_1}/NP_{e_3}
[unique$(e_1)^2$,unique$(e_2)^1$] ==> [unique$(e_1)^1$]
[boy(e_1),gave(e_1,e_2,e_3),girl(e_2)]

The final determiner is incorporated in a similar way:

(47) i. Shift *the*

NP_{e_1}/N_{e_1}
[unique$(e_1)^1$]
[]
NP_{e_1}/NP_{e_3}
[unique$(e_1)^1$]
[boy(e_1),gave(e_1,e_2,e_3),girl(e_2)]

j. Reduce *the-boy-who-gave-the-girl-the*

NP_{e_1}/N_{e_3}
[unique$(e_1)^1$,unique$(e_3)^1$]
[boy(e_1),gave(e_1,e_2,e_3),girl(e_2)]

But now both closure constraints have the same status, both ready to be discharged at the next function application. The application to *present* thus closes both the simple NP *the present* and the larger, complex NP, and since the corresponding closure constraints both have closure points of 1, both are enforced at this stage:

(47) k. Reduce *the-boy-who-gave-the-girl-the-present*

NP_{e_1}

$[unique(e_1)^1, unique(e_3)^1] ==> []$

$[boy(e_1), gave(e_1,e_2,e_3), girl(e_2), present(e_3)]$

This completes our analysis of (45).

This scheme is just one way of implementing the connection between uniqueness constraints and noun phrase closure. Although it is perhaps a little inelegant, it saves us from the alternative of introducing more structure to the syntax or semantics and so keeps the existing representations simple and easy to think about. Besides, unless a given analysis calls for generalised composition, in other examples we will forget about the need for such numerical closure points and use the more rudimentary scheme assumed in section 5.3.3.

5.4. Applications

The model of incremental reference evaluation is now complete, and so it is time to return to the two specific problem areas, outlined in Chapter 1, which we expected to benefit from incremental interpretation. The following section therefore examines the treatment of the troublesome noun phrase *the rabbit in the hat* and considers why it poses no problem for the present system. The subsequent section then returns to Crain and Steedman's and Altmann and Steedman's theory of local ambiguity resolution, and investigates the system's behaviour when faced with a syntactically ambiguous input.

5.4.1. The Rabbit in the Hat

This section illustrates how the model of incremental reference evaluation can account for the problematic noun phrase

(48) the rabbit in the hat

discussed at the beginning of the thesis. The relevant situation pictured in Figure 1.1 may be characterised in the present terms as follows:

(49) {rabbit(r1), rabbit(r2), rabbit(r3), hat(h1), hat(h2), box(b1),
in(r2,h1), in(r3,b1)}

The steps taken in processing the phrase are detailed in Figure 5.10; by way of a summary, we will run through the example in words. (To simplify the exposition, all distinct variables in Figure 5.10 have been assigned different names.)

(a) Shift *the*

NP_{e_1}/N_{e_1}
[unique(e_1)]
[]
[]

(b) Shift *rabbit*

$N_{e_2}/(N_{e_2}\backslash N_{e_2})$
[]
[rabbit(e_2)]
[D_2 : {r1,r2,r3}]
NP_{e_1}/N_{e_1}
[unique(e_1)]
[]
[]

(c) Reduce *the-rabbit*

$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[unique(e_1)]
[rabbit(e_1)]
[D_1 : {r1,r2,r3}]

(d) Shift *in*

$(N_{e_2}\backslash N_{e_2})/NP_{e_3}$
[]
[in(e_2,e_3)]
[D_2 : {r2,r3}, D_3 : {h1,b1}]
$NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[unique(e_1)]
[rabbit(e_1)]
[D_1 : {r1,r2,r3}]

(e) Reduce *the-rabbit-in*

NP_{e_1}/NP_{e_3}
[unique(e_1)]
[rabbit(e_1),in(e_1,e_3)]
[D_1 : {r2,r3}, D_3 : {h1,b1}]

(f) Shift *the*

NP_{e_4}/N_{e_4}
[unique(e_4)]
[]
[]
NP_{e_1}/NP_{e_3}
[unique(e_1)]
[rabbit(e_1),in(e_1,e_3)]
[D_1 : {r2,r3}, D_3 : {h1,b1}]

(g) Reduce *the-rabbit-in-the*

NP_{e_1}/N_{e_3}
[unique(e_1),unique(e_3)]
[rabbit(e_1),in(e_1,e_3)]
[D_1 : {r2,r3}, D_3 : {h1,b1}]

(h) Shift *hat*

N_{e_4}
[]
[hat(e_4)]
[D_4 : {h1,h2}]
NP_{e_1}/N_{e_3}
[unique(e_1),unique(e_3)]
[rabbit(e_1),in(e_1,e_3)]
[D_1 : {r2,r3}, D_3 : {h1,b1}]

(i) Reduce *the-rabbit-in-the-hat*

NP_{e_1}
[unique(e_1),unique(e_3)] ==> []
[rabbit(e_1),in(e_1,e_3),hat(e_3)]
[D_1 : {r2}, D_3 : {h1}]

Figure 5.10. Incremental evaluation of *the rabbit in the hat*.

As Figure 5.10 shows, the first move is to read the initial determiner, consider its lexical entry, and shift the appropriate information onto the empty parsing stack. *The* introduces a closure constraint *unique(e_1)*, meaning that there should be only one value in the domain of e_1 by the time this NP is syntactically closed. The next step, in (b), is also a shift, since reduction requires at least two items on the stack. Shifting the noun *rabbit*, in its type-raised form as a function over noun-modifiers, imposes a constraint of rabbitness

on the variable e_2, and the operation of domain refinement labels this variable with a domain, D_2, consisting of the three rabbits in the context.

A stack reduction can now take place, using the rule of forward composition. This cancels N_{e_1} and N_{e_2}, making *the rabbit* a syntactic function from noun-modifiers to NPs. Assuming that e_1 and e_2 unify to e_1, the predicates *unique* and *rabbit* now apply to the same variable, e_1. The system thus expects the set D_1 to have reduced to a single element by the time the phrase is complete.

In step (d) the preposition *in* leaves the input buffer, introducing two new variables and extensions, e_2 and e_3. As with any new variables, these are initially labelled with a domain containing all entities in the context. Their domains are then refined according the constraint *in*(e_2,e_3). Making D_2 consistent with this constraint eliminates those entities which are not in something; and D_3 is similarly refined, to {h1,b1}, since *h1* and *b1* are the only entities which contain anything. Importantly, the second hat *h2* is not included in the domain of candidates, D_3, from which the inner NP must later take its reference, because it does not contain anything.

The next move, in (e), matches e_1 with e_2, again by the composition rule. The unification of e_1 and e_2 is taken to mean that these variables in fact represent the same extension and so, at the referential level, the corresponding domains D_1 and D_2 are intersected. This eliminates the first rabbit *r1* from D_1, since it is not contained in anything. D_3 is reconsidered in the light of this revision, but both its elements remain consistent with the constraints.

The subsequent shift (f) and reduction (g) with *the* do nothing to further refine the set of containing objects, D_3, but the processor now expects this particular set to be a singleton when the phrase is complete.

The final shift introduces e_4, its constraint *hat*(e_4) determining the set {h1,h2} for the noun. The rule of application is then used for the first time, in (i), closing the noun phrase. Identifying e_3 with e_4 eliminates the box, *b1*, and constrains each candidate in D_1 to be in the remaining hat, *h1*. So *r3* goes out. We are left with two singleton sets, thus ensuring the success of the two checks for uniqueness. The closure constraints *unique*(e_1) and *unique*(e_3) are indeed enforced at this stage, since function application to *hat* has closed both the simple NP (corresponding to e_3) and the complex NP (corresponding to e_1).

Why does our incrementally interpreting processor solve this problem of reference? This is an interesting question, given that it is not captured by many other models, old and

new. For example, Winograd's (1972) program would translate the phrase into the following Planner procedure call:

(50) (FIND 1 ?X s.t.
(PROVE (IS ?X RABBIT))
(PROVE (?X IN ?Y))
(FIND 1 ?Y s.t.
(PROVE (IS ?Y HAT))))

Expressions such as this are evaluated inside-out. First the inner procedure

(FIND 1 ?Y s.t.
(PROVE (IS ?Y HAT)))

is evaluated, and if successful binds ?Y to a (unique) entity; then the outer expression evaluates, with ?Y already bound to this entity. But of course, in the problematic context we have dealt with, Planner would fail to find just one value matching ?Y, so rendering the entire expression in (50) unevaluable.

So why does the present model cope where SHRDLU fell down? Perhaps somewhat surprisingly, the technical reason for the sufficiency of the model comes from the nature of the semantic translations rather than the fact that it performs incremental evaluation. In particular, we have relaxed the compositionality of the semantic translations by separating the constraints of uniqueness from the other constraints. Given that the translations are built up incrementally, separating the uniqueness checks from the other constraints has the effect of giving both determiners scope over the entire noun phrase. The incremental analysis in Figure 5.10 can therefore be seen as a procedural encoding of the following declarative formula:

(51) $(\exists\, !e_1 \in D_1)(\exists\, !e_2 \in D_2)\ \text{rabbit}(e_1)\ \&\ \text{in}(e_1,e_2)\ \&\ \text{hat}(e_2)$

And interpreting the logical form in (51) as a constraint satisfaction problem, any method of evaluating the body of the expression

$\text{rabbit}(e_1)\ \&\ \text{in}(e_1,e_2)\ \&\ \text{hat}(e_2)$

will produce unique values for e_1 and e_2, be it generate-and-test, backtrack search, or, indeed, network consistency. Technically, incremental *evaluation* does not play a key role.

But what originally motivated our structural separation of uniqueness information from the rest, so producing two levels in the semantic translations? Far from being introduced as an *ad hoc* measure to cope with the rabbit phrase, the two-level approach was motivated on the distinct grounds of incremental evaluation. Unlike other lexical items, we didn't want to evaluate determiners immediately, and so they were separated out. So although incremental evaluation is not technically required to solve the problem, it

provides the underlying motivation for the form of the translations, wherein lies the actual solution.

5.4.2. Syntactic Ambiguity

We now consider the kind of guidance given to the syntactic processor by the process of reference evaluation described in this chapter. For this purpose we will continue to assume the simple shift-reduce parser of Chapter 3, augmented with semantic operations. Here we will concentrate on the way in which incremental interpretation can eliminate a developing syntactic analysis, rather than the precise order in which the syntactic alternatives are explored, leaving more general questions about the overall form of the parser until the next chapter. The model's context-dependent predictions for local ambiguity resolution will be analysed in the light of the theories of Crain and Steedman (1985) and Altmann and Steedman (1988), reviewed in Chapter 2. In particular, recall the discussion of Altmann and Steedman's Principle of Referential Support, restated here:

> *The Principle of Referential Support*
>
> An NP analysis which is referentially supported will be favoured over one that is not.

The present model contains two distinct forms of referential support, and either of these may eliminate a developing syntactic and semantic analysis, irrespective of the order in which the alternatives are explored. First, a syntactic step may yield an inconsistent constraint network, deriving constraints which eliminate every element from some value domain. Second, the conditions of uniqueness applied at points of noun phrase closure may not be met, in cases where a value domain contains more than one element. Definite NPs depend on both of these forms of constraint, whereas indefinite NPs only depend on network constraints; in the current formulation, indefinites do not produce closure constraints.

However, we will concentrate on the processing of definite rather than indefinite NPs, since our treatment of indefinites is somewhat unnatural. As we noted in Chapter 1, indefinite NPs more typically introduce new entities to the context, rather than referring to existing elements, as they are assumed to do here. The applicability of the present approach is, at best, limited to those occasions where an indefinite expression is intended to refer to one of a set of entities shared between speaker and hearer. And even when a speaker genuinely intends non-specific reference to an element in a shared set of known entities, say a set of women, he or she is perhaps more likely to say *one of the women* than

simply *a woman*.

Furthermore, even when the indefinite article is used in this way, as in imperatives such as (52),

(52) Pick up a block

the model does not entirely capture the pragmatics of the use of an indefinite article. In particular, if the set of potential referents for an expression is available to both speaker and hearer, then an indefinite article is generally only used when the expression applies to more than one entity. So, if we assume that the reference of the expression in (53)

(53) a block

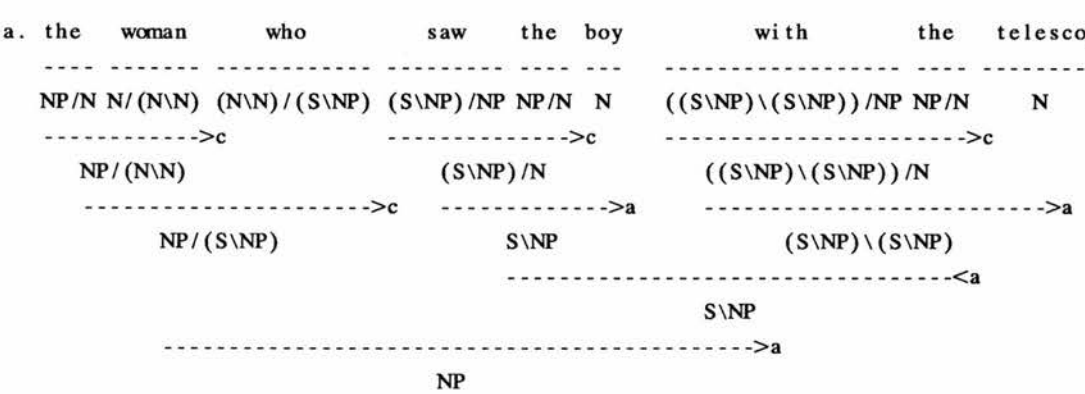
is to be taken from a set of blocks salient in the minds of both speaker and hearer, (53) is appropriate when this set has more than element. In contrast, the definite article is employed if the set is a singleton. A more general model would control for this factor, and regard an indefinite expression as acceptable in this sense only if its value domain is a non-singleton. (One way of doing this would be to associate a closure constraint of *non*-uniqueness with the indefinite article.)

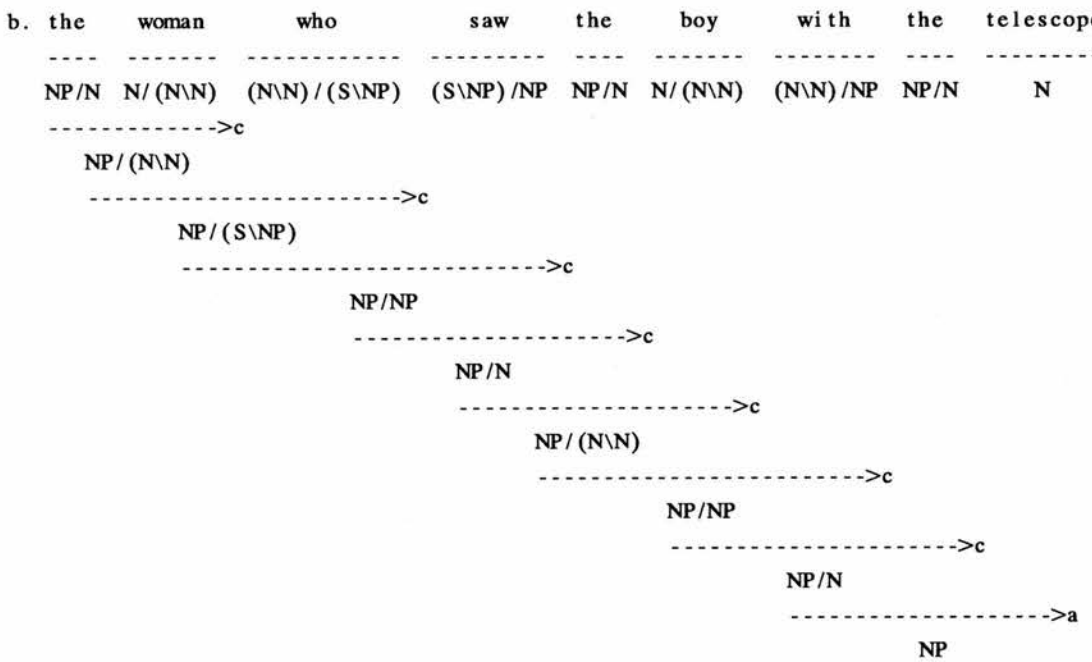
Returning to definites, then, consider the following syntactically ambiguous noun phrase:

(54) the woman who saw the boy with the telescope

In (54), the PP *with the telescope* may attach to either the preceding noun or verb. In the terms of combinatory grammar, the syntactic alternatives are represented by the following two derivations of the string:

(55)





The analysis of (55a), which attaches the PP to the verb, produces the final syntactic/semantic representation in (56a), whereas the noun-modifying reading of (55b) produces the representation of (56b):

(56) a. NP_{e_2}
[]
[woman(e_2),see(e_1,e_2,e_3),boy(e_3),with(e_1,e_4),telescope(e_4)]
[D_1 : {...}, D_2 : {...}, D_3 : {...}, D_4 : {...}]

b. NP_{e_2}
[]
[woman(e_2),see(e_1,e_2,e_3),boy(e_3),with(e_3,e_4),telescope(e_4)]
[D_1 : {...}, D_2 : {...}, D_3 : {...}, D_4 : {...}]

Here the attachment site of the PP is characterised by the distribution of variables within the semantic translations: in (56a) the predicate *with* associates with the variable representing the index of the seeing event, e_1, whereas in (56b) it associates with the variable representing the boy, e_3.

Of course, this simple semantic difference will be reflected in the alternative states of evaluation for (56a) and (56b); and, depending on the context, one or other of these analyses may be inconsistent, and therefore not derivable by the parser. Clearly this behaviour is in accordance with the Principle of Referential Support.

To set the scene, let us first consider the effect of network constraints, and ignore the closure constraints which will be enforced during the analysis of (54). For example, the

context of (57) will prevent the verb-modifying reading of (54), by virtue of the network constraints in (56a). (In order to simplify the exposition, the type of each entity in the context has been omitted. In this and other examples, the reader should assume that {w1,w2,...} are women, {b1,b2,...} are boys, and {t1,t2,...} are telescopes.)

(57) {see(i1,w1,b1), see(i2,w2,b2), see(i3,w3,b3), with(b1,t1), with(b2,t2)}

Here three women {w1,w2,w3} have seen three boys {b1,b2,b3} in three separate events {i1,i2,i3}. The context supports the network constraints of the noun-modifying analysis, in (56b), since two of the boys seen by women are associated with telescopes. In contrast, the context does not support the verb-modifying reading as there are no instances of any seeing event being performed with a telescope.

On other occasions, the process of network consistency alone will not eliminate either analysis. Consider (58), for instance:

(58) {see(i1,w1,b1), see(i2,w2,b2), see(i3,w3,b3),
with(b1,t1), with(b2,t2), with(i2,t3), with(i3,t4)}

In (58), *w1* and *w2* saw boys who were carrying telescopes, and *w2* and *w3* saw boys by using telescopes. In this context, then, network consistency permits both the verb-modifying and noun-modifying readings of the string, their respective analyses configuring as follows:

(59) a. NP_{e_2}
[]
[woman(e_2),see(e_1,e_2,e_3),boy(e_3),with(e_1,e_4),telescope(e_4)]
[D_1 : {i2,i3}, D_2 : {w2,w3}, D_3 : {b2,b3}, D_4 : {t3,t4}]

b. NP_{e_2}
[]
[woman(e_2),see(e_1,e_2,e_3),boy(e_3),with(e_3,e_4),telescope(e_4)]
[D_1 : {i1,i2}, D_2 : {w1,w2}, D_3 : {b1,b2}, D_4 : {t1,t2}]

However, a syntactic analysis must also be compatible with the closure constraints of uniqueness associated with definite determiners. It turns out that the natural interaction of the parsing process with the enforcement of closure constraints captures the essence of Altmann's Principle of Referential Failure, a special case of the Principle of Referential Support.

The Principle of Referential Failure predicts that if a definite referring expression fails to refer to a unique entity, then subsequent material will be treated as modifying the referring expression. To see how this comes about, recall that any noun is form-class

ambiguous between an atomic category N and some type-raised categories which explicitly require one or more modifiers for the noun. Consider just the atomic category and that which is type-raised over one modifier:

(60) a. boy := N_{e_1} : [boy(e_1)]
b. boy := $N_{e_1}/(N_{e_1}\backslash N_{e_1})$: [boy(e_1)]

Given these two definitions for the noun *boy*, and the definition of the definite article in (34), the parser may analyse the fragment *the boy* in two ways. It may combine (34) with (60a) by function application, or compose (34) with the type-raised reading of the noun in (60b). The former analysis, in contrast to the latter, closes the NP and so automatically discharges the closure constraint on e_1, *unique(e_1)*, in parallel with the combination. Thus, if the context contains just one boy, both analyses of *the boy* survive:

(61) a. NP_{e_1}
[unique(e_1)] ==> []
[boy(e_1)]
[D_1 : {b1}]

b. $NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[unique(e_1)]
[boy(e_1)]
[D_1 : {b1}]

In (61a), built by function application, the closure constraint has been successfully discharged. The complex NP reading also survives, though here no attempt is made to satisfy the closure constraint since the NP remains open.

But should the check for uniqueness fail, then the analysis which closes the NP is dropped. The Principle of Referential Failure states that in these circumstances of referential failure, "an analysis which treats subsequent material as [modifying] will be favoured over one that does not." And this is just what happens. If, say, there are two boys in the context, the analysis which persists is that which subcategorises for modifying material, because it does not close the NP and so does not discharge its closure constraint. Thus in cases of referential failure the only reading of the fragment *the boy* is (62):

(62) $NP_{e_1}/(N_{e_1}\backslash N_{e_1})$
[unique(e_1)]
[boy(e_1)]
[D_1 : {b1,b2}]

Hence the procedural effects of closure constraints, introduced to characterise the semantics of definiteness, also interact naturally with the processor to capture the Principle of

Referential Failure.

In order to take a closer look at the predictions which result from closure constraints, we will examine the results of processing (54) in different contexts. The first of these involves two boys:

(63) {see(i1,w1,b1), see(i2,w2,b2), with(b1,t1)}

Here, then, two women each saw a boy, and one of these boys has a telescope. In this context the closure constraint on the set of boys {b1,b2} fails when the simple NP *the boy* is closed, forcing the NP to explicitly subcategorise for a noun-modifier N\N. No matter which route is taken through the syntactic search space, the parser cannot derive an NP by analysing the subsequent PP as a verbal modifier (S\NP)\(S\NP). These results are summarised in (64).

(64) Prediction in context with two referents

PP is NP-attached	POSSIBLE
PP is VP-attached	NOT POSSIBLE

Note that even if the verb-modifying reading was network consistent with the context (i.e. in addition to the facts in (63), some seeing event was done with a telescope), the verb reading would still not be permitted because of the failure of the closure constraint stemming from *the boy*.

For contexts in which the simple NP does determine a unique referent, both readings are potentially accessible to the processor. In the one-referent context, the processor can successfully close the simple NP *the boy* and go on to build an S\NP for *saw the boy*. This enables *with a telescope*, as an (S\NP)\(S\NP), to backwards apply and so produce a VP-attached analysis. However, the success of the closure constraint does not actually prevent the noun-modifier reading, in which *boy* subcategorises for a modifier. As long as *the boy with a telescope* determines a unique referent, the parser will also yield this second analysis. Hence the prediction:

(65) Prediction in context with one referent

PP is NP-attached	POSSIBLE
PP is VP-attached	POSSIBLE

Of course, one or other of these readings may be eliminated by a basic network inconsistency, as we saw above. For example, (66) represents the situation where two women have seen the same boy, one with the aid of a telescope and one without.

(66) {see(i1,w1,b1), see(i2,w2,b1), with(i1,t1)}

Here, then, the simple NP *the boy* refers uniquely, and so both simple and complex

analyses of the NP are pursued after reading and processing *boy*. However, since the context does not directly associate any boys with telescopes, the complex NP analysis *the boy with the telescope* will later be dropped — not because of the failure of a closure constraint, but because there is no known fact which links the boy to the telescope. So in this case, the ordinary, network constraints arising from the string will permit only the VP-attached reading.

The model therefore predicts that attachment ambiguities may often be resolved by reference to the current context. Furthermore, since the referential support for each analysis is computed incrementally, as the syntactic analysis develops, it seems reasonable to suppose that referential information should take priority in guiding the parser in the attachment decision. It would be very odd for structural heuristics such as Minimal Attachment to assume the primary responsibility for disambiguation in a model where the semantic component can provide an immediate contextual assessment of each proposed attachment.

Moreover, the above involvement of definite reference in local ambiguity resolution constitutes an explanation of Altmann's (1986) Principle of Referential Failure. In cases of referential failure, the action of closure constraints, motivated on the quite distinct grounds of noun phrase semantics, interacts automatically with the parser to favour the reading in which subsequent material is treated as modifying the NP in question.

The proposal thus implements an aspect of Altmann and Steedman's (1988) Principle of Referential Support. It is not a complete implementation, though, since Altmann and Steedman argue that in cases of referential *success* the VP-attached analysis presides. Their theory specifies that after reading *boy* the material which follows is treated as noun-modifying only if the context includes more than one boy; otherwise, if *the boy* refers uniquely, the processor discards the noun-modifying analysis and continues with the verb-modifying reading. The Principle of Referential Support thus includes an assumption that the speaker is cooperative: if *the boy* refers uniquely, then the speaker will not complicate the expression with a referentially unnecessary modifier like *with the telescope*, for this would violate Grice's Quantity maxim (cf. Grice, 1975). In contrast, the program will produce both readings in such cases where the simple NP refers successfully, assuming they are both otherwise consistent with the context. However, although this behaviour is inconsistent with A&S's general theory, it is not incompatible with their experimental results; for they report an experiment in which the reading times for one-referent/VP-attached and one-referent/NP-attached sentences were "not significantly different from one another".

It is worth mentioning that nothing of the above compromises the model's ability to parse right-extraposed sentences such as

(67) The man won who gambled

where the main verb *won* separates *the man* from its extraposed, relative clause modifier. (Note, however, that we do not treat the semantics of such main clauses.) Where the simple NP *the man* fails to refer uniquely it will be categorised as an NP/(N\N) — an NP looking for a modifier. Although this modifier does not follow immediately, the rule of crossed backward composition enables the main, intransitive verb to compose, backwards, with the subject, to form a sentence looking for a noun-modifier. The syntax of this rule is restated here for convenience:

(68) Y/Z X\Y ==> X/Z

And the string is derived as follows:

(69)

```
The      man      won        who          gambled
----  -------  ----  ------------  ------
NP/N  N/(N\N)  S\NP  (N\N)/(S\NP)    S\NP
------------>c
   NP/(N\N)
   ----------------<xc
         S/(N\N)
         ------------------------->c
                   S/(S\NP)
                   ------------------------>a
                              S
```

Thus the NP *the man* is not closed until its eventual combination with *who gambled*, to the right of the main verb; accordingly, the closure constraint on the set of men is not discharged until the first invocation of forward application — used to combine *the man won who* with *gambled*.

5.5. Conclusion

The above sections have shown how a process of incremental reference evaluation may act to resolve local and global syntactic ambiguities by failing to interpret contextually inappropriate analyses proposed by a left-to-right parser. This operation has been illustrated in terms of one syntactic type of ambiguity, the ambiguity of PP attachment between verb and noun. But the class of ambiguities resolvable on these grounds is much larger, limited only by the coverage of the grammar and the semantic forms. For example, the fragment of grammar which has been implemented also covers constructions which involve a noun-noun attachment ambiguity (see Appendices A.2 and A.6). This may occur with

either PPs, as in (70a),

(70) a. the door under the balcony with the sign

or with relative clauses, in examples akin to (70b):

(70) b. the door under the balcony which has the sign

Syntactically, the final modifier in both of the above examples may attach to *balcony* or *door*; but pragmatically, the current context may render one of these readings inappropriate. Similarly, the local "analytic" ambiguity in (71),

(71) The woman told the girl that the man knew ...

where *that the man knew ...* may modify *girl* or complement the verb *told*, can be resolved in contextual terms.

Moreover, since references are evaluated incrementally, these ambiguities may be resolved very early in processing. In general, the process of contextual evaluation passes judgement on a developing syntactic analyses after every word is read. We have therefore achieved a far greater degree of incremental interaction between syntax and reference than managed by the programs of Winograd (1972) or Hirst (1987), whose work was discussed in Chapter 2.

The program thus provides the beginnings of a computational model of the HSPM, as envisaged by Crain and Steedman (1985) and Altmann and Steedman (1988). However, although the referential processing described above characterises some significant aspects of Altmann and Steedman's Principle of Referential Support, it does not provide a complete model in several respects. First of all, it has nothing to say about their more general Principle of Parsimony, repeated below.

The Principle of Parsimony

> A reading which carries fewer unsupported presuppositions will be favoured over one that carries more.
>
> (From Altmann and Steedman, 1988)

Crucially, this would involve considering referring expressions which make additions to the discourse context, rather than simply extracting information from it. Here, of course, we are only considering noun phrases which refer to known elements. Secondly, the model does not aim to capture what happens at the point at which the HSPM realises it has taken an incorrect reading of a string. So although we have considered — in computational terms — the reason for an apparent human preference for one reading over another, and just how this preference might arise during incremental processing, we have not considered the recovery mechanism which must be employed in (presumably rare) cases where the

context leads the processor down the garden path. (Of course, in certain extreme instances, such as *The horse raced past the barn fell*, even the recovery mechanism itself appears to fail.)

The next two chapters investigate two residual issues which we have not yet discussed. Chapter 6 considers some ways of managing the syntactic search process, while in Chapter 7 we address the implications for definite reference, contrasting the present approach with that of Mellish.

Chapter 6

Parsing

So far in this thesis we have assumed a very simple model of parsing, the shift-reduce processor introduced in Chapter 3. In Chapter 5 this allowed us to connect the process of reference evaluation to the individual steps of the overseeing parser and investigate the way in which these steps may be blocked by their inappropriateness in the current context. Since a shift-reduce parser may be seen as a standard, canonical form of bottom-up processing (most parsers have the operational equivalents of Shift and Reduce), it has helped demonstrate the role of a parser in general in invoking the incremental process of reference evaluation, without embroiling us in more sophisticated parsing operations of syntactic interest alone.

However, having now dealt with the semantics, the time has come to confront the parsing issue head on. How should a parser explore the syntactic search space determined by combinatory grammar? The first two sections of the chapter point out the issues which are likely to be important for any attempt to answer this question within the present methodology. We first appeal to psycholinguistics, and consider the architectural features which Crain and Steedman argue characterise the structure of the HSPM. We then turn to the computational issues involved, in section 6.2, and analyse the nature of the syntactic ambiguity found in CCG.

With the nature of the problem in mind, we reconsider the shift-reduce processor of Chapter 3. Section 6.3 investigates what happens when a straightforward search is used to explore the tree of options the parser and the grammar determine for an input string, and finds that there is the potential for much duplication of effort. Section 6.4 then demonstrates how this can be avoided by processing the syntax on a chart instead of a stack.

This chapter therefore sets out to review the problems of parsing combinatory grammar and to investigate two regimes for controlling the syntactic search. This will be familiar territory for many readers, but it is important that the issues are clearly stated, especially for the benefit of anyone wishing to extend the model of the previous chapter to incorporate more diverse forms of on-line reference to the context.

6.1. Psychological Considerations

The proposals of Crain and Steedman and Altmann and Steedman for the nature of the HSPM come in two parts. The first part consists of a set of principles purported to guide the parsing process in resolving syntactic ambiguities. We discussed these principles in Chapter 2, and in Chapter 5 saw how the model of reference evaluation gives rise to certain aspects of them (concentrating on the Principles of Referential Support and Failure). But we have so far not mentioned the second part of their theory. This consists of a collection of general architectural properties which they would expect to find in any model of the HSPM which somehow employs their interactive parsing principles. These proposals do not constitute a model of the HSPM, in themselves; rather they are parameters which collectively determine a set of possible interactive processing models.

It is to three of these parameters which we now turn. In each case, we will consider the architectural feature in question and how it relates to the specific, computational model of processing developed in Chapter 5. As we will see, the first two properties, which specify that interaction between syntax and semantics be *weak* and *fine-grained*, have already been dealt with. But the third property, relating to the order in which the syntactic analyser presents its options, has not been addressed. Accordingly, the rest of this chapter examines the parsing issue, in relation to both combinatory grammar and the HSPM. First, then, the three parameters:

Weak Interaction. Crain and Steedman distinguish between two sorts of interaction between syntax and semantics, which they call "strong" and "weak" interaction. The former, strong version allows semantics to actively instruct the syntactic processor as to which grammar rules to apply, whereas the latter, weak version presumes that the syntactic processor has overall responsibility for proposing the alternatives, and semantics merely selects between the alternatives on the basis of their plausibility in the context. They argue on meta-theoretical grounds in favour of weak interaction, by which semantics selects between a set of syntactically determined options.

The computational model already embodies this aspect of Crain and Steedman's general architecture, since, in Chapter 5, we characterised semantic interpretation as a process driven by a syntactic parser. Moreover, the parser continues with a syntactic analysis only if, at each stage, the semantic interpretation associated with the analysis is consistent with the context. Thus, in common with many other programs, weak or "selective" interaction has been implemented by associating semantic operations with each syntactic rule, and

allowing a syntactically-driven manoeuvre to go ahead only if its corresponding semantics can be evaluated.

Fine-Grained Interaction. Crain and Steedman expect the syntactic processor to present the locally available grammatical alternatives very frequently, perhaps on a word-by-word basis. Thus the size of each unit evaluated by the semantic component is typically small, and the resultant interaction with syntax typically "fine-grained".

The computational model already incorporates such intimate interaction between syntax and semantics, since, for many sentences, the rule of function composition allows the developing semantic interpretation to be incremented as each word in the string is read. And as semantic evaluation proceeds incrementally, the syntactic alternatives are semantically reviewed word by word.

Parallel Evaluation. Finally, Crain and Steedman maintain that syntactic alternatives should be explored together, in parallel, rather than in series. This constraint stems from their general Principle of Parsimony, which states that the processor will favour the locally available reading with the fewest unsatisfied presuppositions. If the processor is to decide which local reading has the fewest unsatisfied presuppositions, then the readings must be *compared*; and since the choice is contingent upon a comparison of readings, they must be presented at the same time rather than one after the other. The parser should therefore explore syntactic alternatives in parallel rather than in series.

The ability to explore analyses in parallel is not actually necessary in a processor which chooses between analyses solely in terms of the less general Principle of Referential Support. Referential support can be defined as an all or none property, as it is in this thesis: a syntactic proposal is either referentially consistent or not, irrespective of the consistency of any other developing analyses. Given that this thesis does not aim to implement the more general Principle of Parsimony, we have a degree of freedom in choosing an appropriate syntactic processor. Indeed, the simple non-deterministic shift-reduce parser described in Chapter 3, and assumed since then, suffices for the kind of semantic interaction performed by the program — regardless of the order in which it might explore alternative analyses.

However, as we will see, there are certain reasons for supposing that this simple model of parsing is not the best one. And it is interesting to consider the possible form of parsing models which *do* develop analyses in parallel, even though the limited variety of

reference we have implemented to date does not require it. The next section therefore proceeds with our investigation, concentrating on the computational aspects of the problem.

6.2. Computational Considerations

Although the ultimate goal is to determine a psychologically relevant syntactic processor, much of this chapter adheres to the conventional computer science methodology which accompanies any attempt to identify an appropriate parser for a grammar formalism. For example, we will expect the parser to be sound and complete with respect to the grammar, and designed to explore all syntactic paths rather than just the first acceptable one. This may seem strange in a thesis which claims to be an exercise in computational psycholinguistics. After all, a common approach to the psycholinguistic preference data is to determine a computational architecture for syntactic processing which *fails* to recognise certain readings of globally ambiguous strings. So, the models of Shieber (1983), Pereira (1985), and Wanner (1987) are all deliberately designed to see only one reading of certain globally ambiguous sentences and thus incorporate syntactic preferences. However, our approach is different. Since here we place the primary responsibility for structural ambiguity resolution with semantics rather than syntax, it seems sensible to investigate the matter with a syntactic processor which does not prejudice the issue either way. It is thus appropriate to look for a parser which can parse all sentences derivable from the grammar, and which can produce all readings of an ambiguous input.

In certain cases it also seems reasonable to judge between parsers on grounds of efficiency. Sometimes, of course, we can expect syntactic non-determinism to be drastically constrained by semantic and pragmatic factors, and in these cases we should not be concerned about the complexity of the syntactic process alone. But in other situations discussed below, where we cannot expect semantics to intervene and where all other things are equal, we should strive to find the parser which explores the syntactic search space in an optimal fashion. So if two parsers are architecturally indistinguishable in terms of the general criterion laid down in C&S, but one can take exponential time to parse whereas the other only takes polynomial time, then we should clearly favour the latter.

Having made these general observations, let us now deal with the specifics of our chosen grammatical theory. In particular, what kinds of ambiguity arise in CCG?

First of all, CCG contains *form-class ambiguity*. CCG typically assigns several

categories to many of the words in its categorial lexicon, and collectively these may produce ambiguity in a complete sentence (i.e. global ambiguity) or in its parts (i.e. local ambiguity). For example, the word *that* has a number of categories, some of which are:

(1) that := S'/S
:= (N\N)/(S\NP)
:= (N\N)/(S/NP)

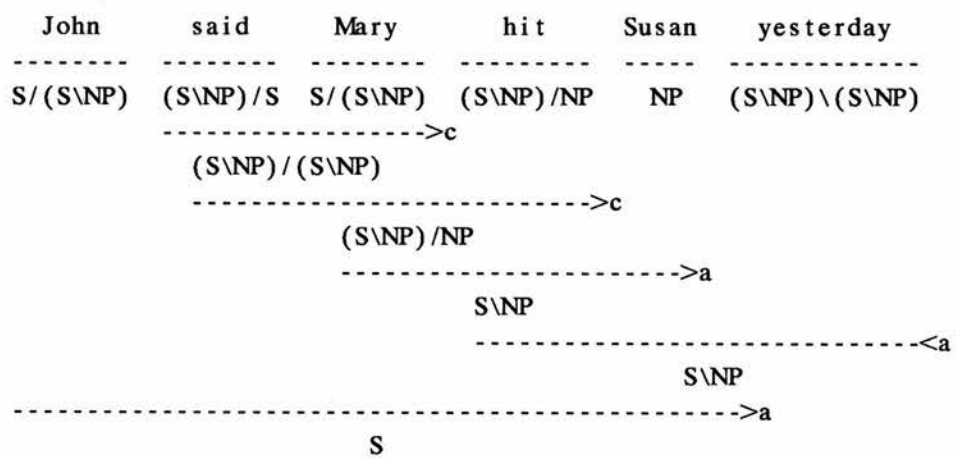
The first category in (1), S'/S, is appropriate where *that* is a complementiser: for example, as in *The tourists told the guide that they enjoyed the tour*. The latter two categories characterise *that* as a subject relative pronoun and object relative pronoun respectively.

Second, CCG contains *structural ambiguity*. This occurs where alternate sequences of rule applications to lexically unambiguous material express genuine ambiguities of scope or attachment. Take the globally ambiguous sentence in (2), for example, in which *yesterday* may be read as modifying the saying or the hitting.

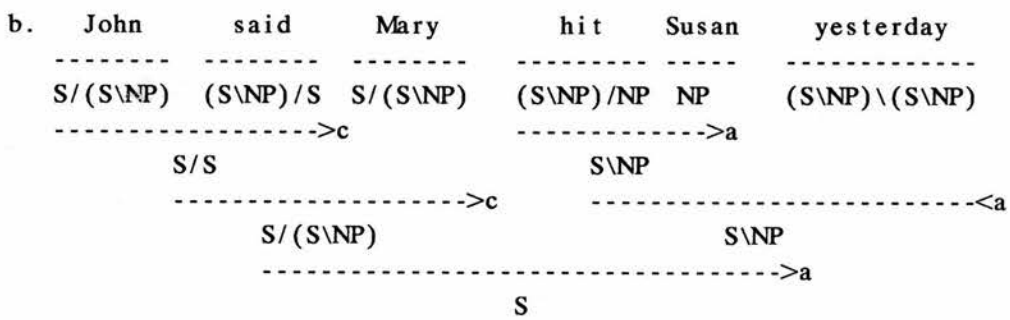
(2) John said Mary hit Susan yesterday

The derivation in (3a) represents the former reading, building an explicit constituent for the finite verb phrase *said Mary hit Susan* before applying the adverbial modifier backwards.

(3) a.



b.



By contrast, an analysis representing the latter reading, such as that of (3b), builds a separate VP for *hit Susan* and modifies this VP with *yesterday*, indicating that Susan was hit yesterday.

The reader should be familiar with the notions of form-class and structural ambiguity, since they can be found in most grammars. However, CCG also contains a third kind of ambiguity, which is not so widespread among grammars. This is what Wittenburg (1986) calls *spurious ambiguity*. Spurious ambiguity arises when syntactically distinct derivations of a string are semantically equivalent. We touched on this issue in Chapter 3, where we noted that such semantic equivalence emerges from the associativity of the rule of function composition.

Let us consider this point again, in the abstract. Consider a string of the form in (4):

(4) X/Y Y/Z Z

There are two ways of deriving the category X from those in (4). The first of these, in (5a), composes from the left and gives (4) a left-branching syntactic structure.

```
(5)  a.  X/Y  Y/Z  Z
         --------->c
           X/Z
           ---------->a
                X
```

The second analysis avoids function composition altogether, implying a right-branching structure:

```
(5)  b.  X/Y  Y/Z  Z
                 ------>a
                    Y
           --------->a
              X
```

Now suppose that the category X/Y has the semantic interpretation x, Y/Z has interpretation y, and Z has z. Using juxtaposition to signify function application and the symbol $\bigcirc$ to signify function composition, (5a) yields the interpretation in (6a), while (5b) produces (6b):

(6) a. $((x \bigcirc y)\ z)$
    b. $(x\ (y\ z))$

But the associativity of function composition amounts to the following equivalence, for any x, y and z:

(7) $((x \bigcirc y)\ z) \equiv (x\ (y\ z))$

So the two analyses in (5) are semantically identical. Notice that we came across an

instance of this phenomenon in 3.6, when considering the two syntactic derivations of *John eats apples*.

In summary, CCG contains three kinds of ambiguity: form-class ambiguity, structural ambiguity, and spurious ambiguity. The first two are genuine, whereas the third, as its name suggests, is not.

At this point we should recall the parsing criteria we set for our own psycholinguistic investigation. We want our parser to be sound and complete, and to develop all analyses in parallel; this way, semantics can do all the choosing. Clearly, then, we require the parser to enumerate and explore all the possible form-class and structural ambiguities in parallel, as the input string is read from left to right. But what of the spurious ambiguity? Here it seems implausible to explore all the syntactically distinct derivations in the hope that semantic factors will choose between them, because they are, by definition, semantically equivalent. A more promising approach would be to explore just one of the set of semantically equivalent analyses — say, the most "incremental" analysis.

In passing, we should note that other aspects of processing might in fact allow the HSPM to choose between the developing syntactically distinct but semantically equivalent analyses. In particular, Steedman (forthcoming) points to the intonation contour of the speech stream as a possible source of guidance for syntactic processing. Related to the intonation contour is the pragmatic, discourse-level given/new structure of the sentence (cf. Halliday, 1967), and so this too might indicate exactly which syntactic constituents should be formed by the parser.

It is also worth pointing out that the process of reference evaluation described in Chapter 5 can, on occasions, eliminate some of the spurious ambiguity. For instance, recall the processing of *the rabbit in the hat*, in its problematic context, as discussed in 5.4.1. Given the available rules of nominal type-raising and function composition, this string has a good many syntactically distinct derivations. All of them are identical in terms of the corresponding pure, strictly compositional semantics, but the procedural effects of the on-line reference process will prevent a number of the derivations developing. In the particular two-hat context considered in 5.4.1, the enforcement of the closure constraint on the set of hats will block all those derivations which form a separate NP category for *the hat*, since *the hat* does not refer uniquely in isolation from the preceding material.

Thus the on-line referential results of the various syntactically distinct readings are not always equivalent. However, it is hypothesised that no "wrong" referential analyses will be introduced as a result of the spurious ambiguity. In particular, the mechanism

linking closure constraints to function application should behave consistently under the non-incremental analyses, firing a uniqueness condition whenever a definite NP is closed. So we do not expect any of the non-incremental readings to lead to referentially improper results — for example, where a definite NP is accepted even though the available constraints do not determine a singleton set of referents. It is unclear whether this hypothesis would still hold if the same framework is extended to cover other classes of NP, for instance those involving indefiniteness or negation.

The spurious ambiguity problem is currently a very active area of research. We will review some of the work in this area in section 6.4, but until then we will have no more to say about it, and the reader is referred to work by Wittenburg (1986, 1987), Pareschi and Steedman (1987), Hepple (1987), Morrill (1988), and Moortgat (1988) for further discussion. It should be stressed that both of the parsers discussed below are naive with respect to this problem: as far as these models are concerned, there are just two sorts of ambiguity in the grammar — that arising from the lexicon and that arising from the combinatory rules. For the purposes of the following sections, then, we will lump genuine, structural ambiguity and spurious ambiguity together, under the name *combinatory ambiguity*. And since type-raising (which may also produce spurious ambiguity) is formulated as a lexical operation, the options which it produces are simply seen as an instance of form-class ambiguity.

6.3. Shift-Reduce Parsing

Let us now return to the simple non-deterministic shift-reduce parser, described in Chapter 3 and assumed since then. The shift-reduce model embodies three forms of non-determinism, and these are related to the grammatical ambiguity as follows.

Form-class ambiguity is directly expressed by shift-shift non-determinism in the parser: if the next word in the input string is multiply ambiguous, the parser has a choice of categories to shift onto the stack. Combinatory ambiguity (i.e. both structural and spurious ambiguity), which arises from different sequences of rule invocations, is captured by the parser's reduce-shift non-determinism. Subject to the present state of the stack and buffer, at each point the parser has the choice of reducing the topmost elements of the stack in accordance with some combinatory rule, or leaving these elements uncombined and making another shift. The alternative routes through the reduce-shift space naturally correspond to the different orders of rule applications which are possible in analysing a given string.

Finally, reduce-reduce non-determinism allows for a specific kind of combinatory ambiguity; where more than one combinatory rule matches two particular adjacent categories. In our particular grammar the combinatory rules appear to be mutually exclusive, and so we will not focus on reduce-reduce non-determinism in subsequent examples. However, all CCG parsers must make provision for it, since it may occur in principle. Consider, for instance, two categories of the following form:

X/X (X/X)\(X/X)

These two categories can reduce either by backward application, or by a slash-mixing form of generalised forward composition. (Note that it is the latter of these which is not included in our rule set.)

Up to now we have been assuming that the parser just happens to resolve shift-shift and reduce-shift uncertainties in a manner appropriate for the analysis of the sentence as a whole; we have ignored the form-classes and derivations which are inappropriate for the task in hand. Of course, weak interaction with the semantic processing of Chapter 5 will typically eliminate some readings as they develop, but how should the parser explore the syntactic alternatives in the first place?

Given that shift-shift, reduce-shift and reduce-reduce non-determinism may all give rise to genuine ambiguity, the entire search space formed by these three dimensions of choice must be explored. And C&S's criterion of parallel evaluation suggests that this should be done in a breadth-first, pseudo-parallel fashion. Such a search process can be implemented, in the usual way, by maintaining a set of stack/buffer configurations. At each step of the parsing process, this set is developed by applying reduce or shift operations to each of its elements in turn.

Consider how this works with respect to reduce-shift non-determinism: at every stage, each configuration in the set is replaced by two new configurations (when both reduce and shift can independently apply to the original), or one new configuration (when only one of reduce or shift can apply), or zero new configurations (when neither can apply). Moreover, if a shift can apply, the configuration spawns n new shifted configurations, where the next word in its input buffer is n ways form-class ambiguous. Similarly, if a reduction can apply to the top two elements, the configuration yields n new reduced configurations, where n is the number of possible reductions.

This is a straightforward but unintelligent method of search, involving much duplicated work. For instance, consider the kind of reduce-shift search which arises from the

following string, assuming the lexical categories below:

(8)	John	will	believe	that	I	said	I	liked	him
	S/(S\NP)	(S\NP)/VP	VP/S'	S'/S	S/(S\NP)	(S\NP)/S	S/(S\NP)	(S\NP)/NP	NP

Ignoring any form-class ambiguity, there are 1,430 syntactically distinct analyses of the string in (8). (Such figures can be derived from the series of Catalan numbers, as Wittenburg (1986) has shown.) On the assumption of section 6.2, that the parser explores all syntactically distinct readings regardless of any spurious ambiguity, we expect our parser to produce each of these analyses. The problem is that the simple non-deterministic shift-reduce parser will produce them in a very inefficient manner.

To see why, let us concentrate on a particular class of analyses of (8): those which form an S/S' for *John will believe*, and an S' for *that I said I liked him*, before combining these two categories by forward application to produce an S for the complete sentence. Given the spurious ambiguity, the substring *John will believe* can be analysed in two ways, and the substring *that I said I liked him* in 42. But the search for all 42 analyses of this latter substring will be performed *twice*, in exactly the same manner: once for each analysis of *John will believe*. Thus the reduce-shift search tree contains two identical subtrees representing the search for all analyses of *that I said I liked him*. Similarly, the space contains five identical search subtrees for *I said I liked him*, and so on. Equivalent duplicate behaviour stems from the shift-shift and reduce-reduce processing, although the algorithm can be easily amended to cope more intelligently with these latter two forms of non-determinism.[1] This is a widely acknowledged problem in such parsers, and the standard remedy is to use a chart parser.

6.4. Chart Parsing

Chart parsers offer an alternative approach to parsing combinatory grammar, specifying syntactic processing in terms of a well-formed-substring table or *chart* (cf. Kay, 1980), rather than a shift-reduce stack.[2] The following algorithm takes after Calder's chart-based procedure for parsing Unification Categorial Grammar (Calder *et al*, 1986; Calder, Klein

[1] Form-class ambiguity may processed in a more efficient fashion by shifting *sets* of lexical categories onto each stack, and re-defining the reduction procedure appropriately. I have written a shift-reduce parser exploiting this technique, which is similar to Allen's (1987, pp172-176) deterministic simulation of a non-deterministic finite state automaton, but it offers no remedy for the problem of duplicating subtrees in the reduce-shift search space. For more details, see Trehan and Wilk (1987), who implement the method in the parallel programming language Parlog.

[2] Tomita (1987) presents another alternative, by incorporating various structure-sharing operations into a shift-reduce parser; he applies the model to CCG in Tomita (1988).

and Zeevat, 1988), building constituents bottom-up. The complete set of analyses is explored breadth-first, even though with combinatory grammar these will often be semantically equivalent and not represent genuine ambiguity.

The breadth-first algorithm of Figure 6.1 activates edge-building operations whenever a word is read. Scanning from left to right, the algorithm makes as many combinations as possible after each word is read. Thus, for each form-class for an input word, it forms a new edge corresponding to the location of this word in the string and "labels" the edge with the category in question. It then tries to Reduce this new edge with any existing edges which sit to its immediate left, by matching their respective labels against a combinatory rule. If a rule sanctions a reduction, a new edge is formed to represent the new, reduced constituent, and the parser recursively Reduces with respect to the freshly added edge.[3]

Notice that instead of shift-shift non-determinism, the algorithm contains the loop at step (2.1) of Chart Parse. The effect of reduce-shift non-determinism is captured by the loop at step (4) in Reduce, while the loop at step (4.1) caters for any reduce-reduce

---

Chart Parse. Do steps (1)-(3):

(1) Initialise vertex counter $j \leftarrow 0$.

(2) Reading from left to right, for each word in the input string do steps (2.1)-(2.2):

  (2.1) For each category C for word do steps (2.1.1)-(2.1.2):

    (2.1.1) Add an edge E spanning $(j, j+1)$, labelled C.

    (2.1.2) Reduce w.r.t. E.

  (2.2) Reset vertex counter $j \leftarrow j + 1$.

(3) If there is an edge spanning $(0, j)$ and labelled S, then report success; else Fail.

To **Reduce** w.r.t. an edge spanning (j, k), labelled B, do step (4):

(4) For each edge spanning (i, j), labelled A, do step (4.1):

  (4.1) For each rule s.t. A B ==> C, do steps (4.1.1)-(4.1.2):

    (4.1.1) Add an edge E spanning (i, k), labelled C.

    (4.1.2) Reduce w.r.t. E.

Figure 6.1. A breadth-first chart parser.

---

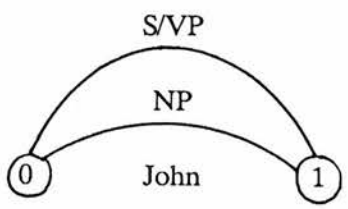
choices.

By way of an example, consider how the chart develops in processing the string in (9):

(9) John believes Mary ...

Assuming that *John* is categorised as both an atomic NP and as a type-raised subject (which we will abbreviate as S/VP), reading the first word causes two edges to be added to the chart:

(10) a.

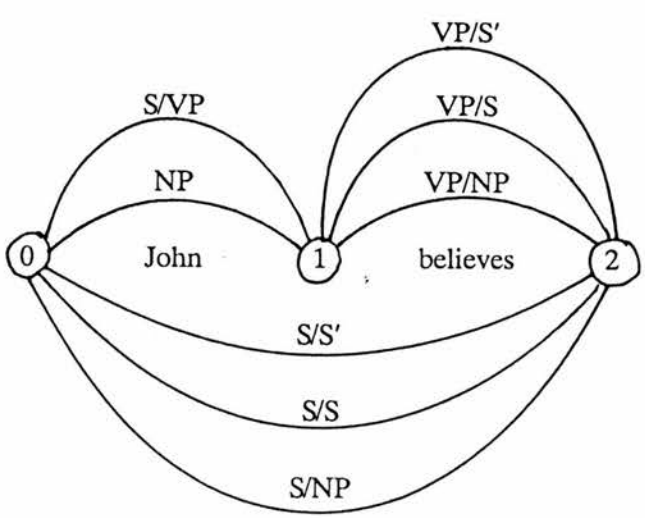


Three plausible categories for *believes* are those in (11), again abbreviating S\NP as VP.

(11) believes := VP/NP
:= VP/S
:= VP/S'

These definitions permit the verb three different syntactic types of object, corresponding to a noun phrase, a sentence and a complementised sentence, respectively. Given the lexical options, the next step produces the configuration in (10b):

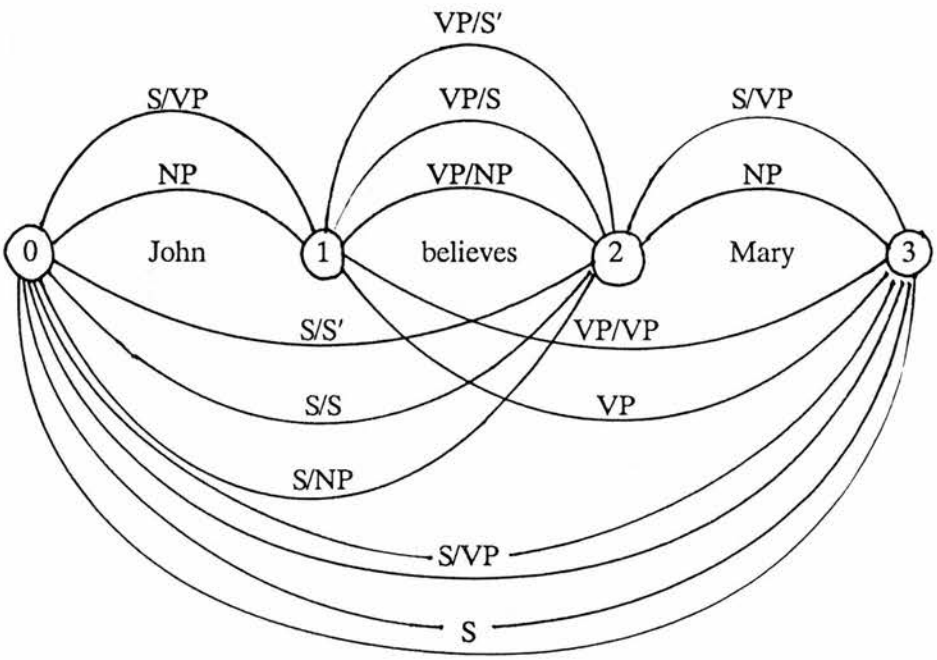
(10) b.



Note that all of the verbal categories combine with the type-raised category for *John* but none combine with the alternative, simple NP category. Finally, two edges corresponding to *Mary* are added to the chart. These can combine with some of the edges for *John*

believes, producing two edges which span all three words: one labelled S and the other labelled S/VP. However, the edges for *Mary* may also combine with the lexical edges for *believes*, and these may in turn reduce with the sentential subject:

(10) c.



As (10c) shows, this allows the two edges spanning (0, 3), labelled S and S/VP respectively, to be derived in two ways; as indeed we should expect from an all-paths parser.

Although the algorithm of Figure 6.1 is presented as a sentence recogniser, in this project it is used to recognise noun phrases and drive the semantic and contextual processes described in Chapter 5. So at step (2.1.1) of Chart Parse, each lexical edge is labelled with a semantic translation and contextual constraint network, in the manner of the referentially evaluating Shift operation developed in the earlier chapter (see Figure 5.8). Just like our previous stack entries, then, an edge label has four parts: a category, a set of closure constraints, a set of network constraints, and the network itself. Accordingly, step (4.1.1) of Reduce is furnished with the semantic operations of the stack-based reduction mechanism (see Figure 5.9). And weak interaction between syntax and semantics is implemented in the obvious way, by making edge additions contingent upon network consistency and, where appropriate, the successful enforcement of closure constraints. (For the interested reader, Appendix A.4 presents the Prolog code for the chart parser and its interface to the semantic evaluation procedures.)

---

[3] In Figure 6.1 an edge is said to span the vertices (m,n) on the assumption that $m < n$.

Ignoring spurious ambiguity, the advantage of the chart is that it never makes the same parsing steps twice, a fact which can be verified by observing that all parsing steps are invoked by additions to the chart. This was not the case with the simple non-deterministic shift-reduce algorithm of the previous section. For example, given a genuine form-class ambiguity in some prefix of the input string, the shift-reduce process would perform a full search of analyses for the suffix more than once.

However, the existence of spurious ambiguity in the grammar will lead to the addition of chart edges with identical labels; this happened in (10c), for example. To prevent this redundancy, we can employ a well-known check, which involves making sure that syntactically and semantically identical edges are not added to the chart. In the implemented system, whenever a new edge is proposed between two vertices, the parser compares its category and semantic translation with those of any existing edges already spanning the same vertices. If the edges' syntactic components are identical, and if the semantic parts of the label are functionally equivalent (i.e. if the predicates match and the semantic variables distribute in the same way), then the proposed edge is discarded.

In (10c), this technique will prevent the installation of the two duplicate edges spanning (0,3) and labelled S and S/VP. However, it can do nothing to stop the prior installation of their "daughter" edges which span (1,3) — so although it prevents the *presentation* of semantically equivalent derivations, in general it will not stop the *search* which produces them.

Recent work by Hepple and Morrill (reported in Morrill, 1988) has formalised the notion of spurious ambiguity in generalised categorial grammars. By defining equivalence classes of derivations, they show how a chart parser can exploit knowledge about the nature of the grammar to prevent spurious edges being added without the need for a "brute-force" comparison of edges. Pareschi and Steedman (1987) also directly confront the problem of developing equivalent readings, by designing a parser which tries to restrict its search to just those paths which will ultimately produce genuinely different sentential analyses. They do this by taking a chart-based algorithm equivalent to that of Figure 6.1, and attempting to constrain its breadth-first syntactic search with an operation called "right-generator marking". The result is a parser which explores form-class ambiguity in parallel, but resolves combinatory ambiguities by reducing as much as possible when reading from left to right. However, Hepple (1987) has closely examined Pareschi and Steedman's technique of right-generator marking and demonstrates that it is incomplete with respect to the grammar. He provides a number of grammatically well-formed sentences which the parser

fails to recognise.

6.5. Summary

We have seen how combinatory grammar contains three distinct forms of syntactic ambiguity: form-class ambiguity, structural ambiguity, and spurious ambiguity. We have considered two parsers which do not distinguish between the genuine and spurious ambiguities in a string, and concluded, in line with other projects using categorial grammar, that the search may be organised profitably in terms of a chart. Since the chart parser in question reads from left to right and explores analyses in pseudo-parallel, it might also be adaptable to the interaction with semantics implied by the Principle of Parsimony — though naturally this is an issue for further research.

Chapter 7

Conclusions

There are two main points to this concluding chapter. First, we return to the problem of timing the evaluation of definite noun phrases: at which point in sentence processing should we expect a definite noun phrase to refer uniquely? This subject was first broached in Chapter 2, when discussing Mellish's and Ritchie's claim that reference may not necessarily be resolved on local grounds. We encountered the issue again in Chapter 5, where uniqueness checks were formulated as constraints to be enforced at points of syntactic closure. In section 7.1, we contemplate the implications of the present approach and contrast them with the predictions of Mellish's model. In the light of the data we will see that the issue is complex and deserves further investigation. This brings us to the second purpose of the chapter, undertaken in section 7.2, which is to summarise some possibly fruitful avenues for further research.

7.1. Implications for the Timing of Definite Reference Evaluation

The proposed treatment of singular definite noun phrases embodies some important claims about the nature of the reference process which we should investigate. In particular, the approach taken to

(1) the rabbit in the hat

suggests that a definite reference may be constrained by *preceding* referential information. Here, the reference of *the hat* is affected by the referential information directly arising from *the rabbit in*. But since a condition of uniqueness is expected to hold when a definite expression is syntactically closed, the model also embodies the claim that a definite reference may *not* be constrained by *subsequent* referential information. Are these predictions borne out by the data? We will first consider this question in the context of definite NPs embedded in larger, complex NPs, akin to (1), and then turn to the putative affect of the enclosing sentence on any definite NP arguments.

7.1.1. The Noun Phrase Level

Consider the following relativised NP:

(2) the hat that contains the rabbit

In the familiar two-rabbit/two-hat context pictured in Figure 1.1, the expression in (2) seems to pick out *hat1* without any trouble, and yet the embedded definite *the rabbit* fails to refer uniquely when considered in isolation. In (2), then, the reference of the embedded NP appears to be constrained by referential information derived from the enclosing complex noun phrase, just as it does in (1).

The computational model predicts the felicity of (2) in this context, deriving a unique referent for the rabbit and the hat. It also predicts that in the context pictured in Figure 7.1 involving an arrangement of blocks, boxes and tables, the expression in (3) can be used to refer to *block1*. (The prepositional phrase *on the table* should be read as modifying *box*.)

(3) the block in the box on the table

Here, neither the complex NP *the box on the table* nor the simple NP *the table* embedded in (3) refer successfully when considered in isolation from the preceding material; each is dependent on what has gone before for unique reference. And yet again the phrase sounds natural in referring to the object in question.

So the prediction that an embedded NP can be referentially "dependent" on referential information derived from the preceding part of the enclosing NP seems to be a correct one. What about the prediction that a felicitous noun phrase refers uniquely at the point of closure, and so is referentially *independent* of the material that follows?

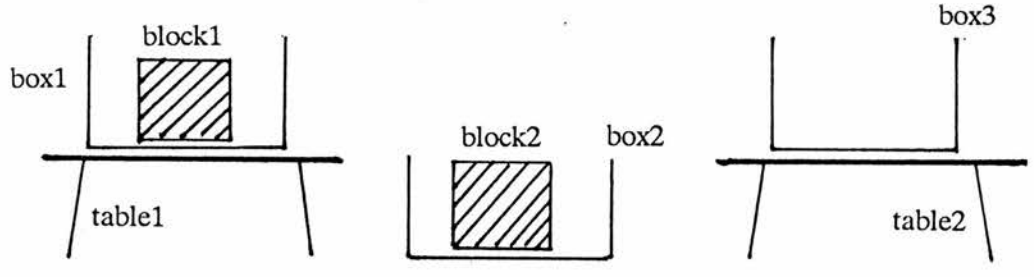


Figure 7.1. Context for *the block in the box on the table*.

Again, let us consider this question where one NP is embedded inside another. For example, the model predicts that the reference of *the hat* in the following object relative correlate of (2)

(4) the rabbit that the hat contains

must be independent of the constraint that it should contain a rabbit. As can be seen from the derivation of (4) in (5) below, the embedded NP *the hat* can only combine with the surrounding material once it is syntactically closed:

```
(5)  the    rabbit      that          the          hat   contains
     ----   -------  ------------  ------------   ---   ---------
     NP/N   N/(N\N)  (N\N)/(S/NP)  (S/(S\NP))/N   N     (S\NP)/NP
     ------------>c                ---------------->a
        NP/(N\N)                      S/(S\NP)
        ------------------------>c    ------------------------>c
                NP/(S/NP)                        S/NP
                ------------------------------------->a
                                NP
```

Since uniqueness is expected on closure, *the hat* must always refer successfully in isolation if the complex NP as a whole is to be accepted. Interestingly, this object relative does sound rather odd in the context of Figure 1.1, where there are two hats. In contrast it seems quite acceptable in a one-hat context.

However, consider the context in (6):

(6) {rabbit(r1), rabbit(r2), rabbit(r3), hat(h1), hat(h2),
in(r1,h1), in(r2,h2), with-big-ears(r1), with-big-ears(r3)}

So we have three rabbits and two hats, each of the two hats containing a single rabbit. One of the contained rabbits has big ears, as does the rabbit not in any hat. Now consider the expression:[1]

(7) the rabbit in the hat with big ears

The acceptability of (7) is not so clear cut. The program predicts that (7) is infelicitous, because *with big ears* may modify *rabbit* only by first closing the embedded NP *the hat*. (Though note that plural NPs like *big ears* are not treated.) But since there are two hats containing rabbits, it cannot successfully make this closure. Thus the only reading available to the parser is that which treats *with big ears* as a modifier of *hat*, and as there are no hats with big ears in the context, it fails to find a reading for the NP as a whole.

[1] I am grateful to David Carter for this example.

Our intuitions do not help much when judging such intricate examples. It may be that in processing (7), we first try to attach *with big ...* to *hat* and, when that reading fails, relax the closure constraint on *the hat* and attach the modifier to *rabbit*; this reading would be ultimately acceptable since it produces unique referents for the definite expressions in the phrase. Or it may be that the connection between noun phrase closure and uniqueness needs to be relaxed; as Mellish (1985) indeed advocates.

7.1.2. The Sentence Level

At this point, it is worth considering the putative affect of the enclosing sentence on the reference of its subject and object NPs. When considering main clauses in this respect, however, we must be careful to distinguish between the informational states of the speaker and hearer, since some of the information conveyed by a sentence will typically be new to the hearer. And new information, by definition, cannot provide referential constraints for the hearer in the sense of the above discussion (though it may provide other kinds of constraints, as we will discuss later). So whereas we assumed that the same contextual information was available to both speaker and hearer when discussing given, definite NPs, we will now consider situations where their information states differ and where the purpose of an utterance is to reduce this informational imbalance.

For example, the speaker and hearer may have the following differing views of a context involving one rabbit and one hat:

(8) A's model of the context: {hat(h1), rabbit(r1), in(r1,h1)}
B's model of the context: {hat(h1), rabbit(r1)}

In these circumstances the speaker A might use the declarative in (9)

(9) A: The hat contains the rabbit

to inform the hearer B of the fact *in(r1,h1)*. Naturally enough, in such situations where the main verb presents the hearer with new information, local, referential knowledge derived from the verb cannot constrain the reference of its argument NPs because at the time of utterance the hearer is unaware of this information. So rather than checking that the containment relation holds between the available hat and rabbit in his or her model of the context, the hearer instead updates the model with the appropriate relationship.

Let us make a brief diversion to see how this aspect of the hearer's comprehension process might be captured in the program as it stands. Our concern will be with how new information could be represented, and how it would interact with the existing processes for

reference evaluation. We will not address the highly complex and logically distinct issue of how a program should decide just which information is given and which is new, and instead assume that it can tell which is which by one means or other.

We could represent the distinction between new and given information by adding another list of predicates to the semantic description of a constituent, separate to the lists of constraints and closure constraints arising from a given noun phrase. Suppose, as we do above, that in processing (9) in the context of (8), the hearer (or, equivalently, the program) regards the new information to be the containment relationship between the hat and the rabbit. The representation of the fragment *The hat contains the*, syntactically an S/N, would then be:

(10) The hat contains the ...

S/N_{e_2}

$[unique(e_2)]$	*Closure Constraints*
$[hat(e_1)] : [in(e_2,e_1)]$	*Given : New*
$[D_1 : \{h1\}]$	*Extensions*

(To simplify matters, throughout this discussion we ignore the additional symbol representing the particular event or state denoted by the verb.) The first list in (10), the closure list, contains just the closure constraint arising from the second determiner in the sentence; the first closure constraint would have been successfully discharged after reading *hat*. The given information so far derived from the sentence is represented below, on the left of the colon; in this case, the single constraint $hat(e_1)$. As before, the contextual extensions are derived solely from this list, here producing a single domain containing the entity *h1*. The list on the right stores the new information that e_1 contains e_2. By the end of sentence processing, the object NP *the rabbit* will have determined a unique rabbit, discharging its closure constraint, and the representations will configure as follows:

(11) S

$[]$

$[hat(e_1),rabbit(e_2)] : [in(e_2,e_1)]$

$[D_1 : \{h1\}, D_2 : \{r1\}]$

And at some point during sentence processing, the predicate $in(e_2,e_1)$ must be added to the program's view of the context, along with assertions to the effect that $e_1 = h1$ and $e_2 = r1$.

In this approach, then, it is clear that the predicate $in(e_2,e_1)$ does not referentially restrict the extensions of either the subject or object NP. The predicate is not input to the consistency procedure, instead being regarded as information to add to the context. The domains D_1 and D_2 are thus restricted only by their associated unary predicates and, as far

as the consistency procedure is concerned, are not connected in any way. (The resultant constraint network is therefore structured as a *forest*: the disjoint union of two single-node tree-structured networks.)

So where the main verb of a sentence introduces new information, it cannot referentially constrain the extensions of any given NPs embedded in the sentence. But what about sentences which relate to information already known to the hearer? Take interrogatives, for instance, typically employed when (the speaker believes) the hearer knows more than the speaker. If A and B are back in the state indicated in (8), agent B may ask:

(12) B: Does the hat contain the rabbit?

To answer this question, the hearer A must verify that a hat contains a rabbit. In addition, the hat and the rabbit are expected to be unique. This task of verifying the proposition *the hat contains the rabbit* seems similar to the task of establishing the reference of *the hat that contains the rabbit*; both would appear to require a process of matching some predicates against some known facts about the context. At a first glance, then, it looks as if a yes/no question such as (12) should direct the semantic processor to interpret all information as given, posting the three constraints

$hat(e_1)$
$in(e_2,e_1)$
$rabbit(e_2)$

to the consistency algorithm to determine if they can be satisfied.

However, there is an important difference between the process of verification employed to answer a question and the process of reference evaluation; for whereas in (2) the reference of *the rabbit* may be constrained by the preceding material, in (12) it cannot. Suppose that B is in fact mistaken in believing there is only one rabbit and, as A knows, there are actually two:

(13) A's model of the context: {hat(h1), rabbit(r1), rabbit(r2), in(r1,h1)}
B's model of the context: {hat(h1), rabbit(r1)}

If the speaker B now puts the question in (12) to the hearer A, we would expect the response to be something like *Wait a minute, there are two rabbits* and so signal the speaker's inappropriate use of the definite article. The hearer recognises that the definite article has been used infelicitously, because from his or her point of view the noun phrase *the rabbit* does not refer uniquely. But if the hearer's process of verifying the proposition *the hat contains the rabbit* were based on the kind of incremental constraint satisfaction used for reference evaluation, this infelicity would not be noticed. Satisfying the

constraints

$hat(e_1)$
$in(e_2,e_1)$

yields domains

D_1 : {h1}
D_2 : {r1}

and so by the time the object NP is reached, *the rabbit* would refer to a unique rabbit. Hence an account of yes/no questions in terms of exactly the same mechanisms used for reference would make the wrong predictions, and thus appears to be incorrect.

In fact this is what we should expect, for although the hearer A knows everything required to answer the question in (12), the relevant facts are not shared between the hearer and speaker. The fact that B asks A whether or not the hat contains the rabbit tells us that the certain facts about containment are not shared; that is, they are not *mutually* believed or known (cf. Clark and Marshall, 1981). And because the containment relation is not mutually known, it seems likely that the hearer does not interpret the predicate $in(e_2,e_1)$ as a "referential" constraint, and so our program should not post it to the consistency algorithm along with $hat(e_1)$ and $rabbit(e_2)$. Instead it is marked as "non-mutually-known" and applies in some other way. In contrast, the definite referring expressions we consider do refer to information shared between speaker and hearer, and so within such expressions NPs can affect each others' reference, as we have seen.[2]

The above data therefore suggests that the reference of a definite noun phrase is not a function of any referential information known to the hearer which is derivable from the enclosing sentence. It thus seems that the province of "referential dependency" such as reported at the beginning of this section is limited to the noun phrase, where, in certain cases, the reference of an NP is constrained by the information provided within a larger, enclosing NP.

These facts warrant a closer investigation into Mellish's claim that a definite noun phrase does not have to refer uniquely at the point of closure. Recall that Mellish argues that a variety of constraints from the preceding and subsequent discourse, the enclosing sentence, and the noun phrase itself may restrict its reference. In particular, he provides a number of examples in which general knowledge about the physical world is claimed to

[2] An alternative way of explaining the inappropriateness of (12) in the context of (13) might be to impose a constraint on the use of function composition, preventing composition into the object NP in main clause cases like (12). In (12) this would force *the rabbit* to be evaluated in isolation of the preceding material, and so capture the unacceptability of the expression in contexts like (13).

affect the reference of a definite NP. For instance, he shows how the set of available referents for *the particle* in

(14) The particle moves from A to B with a velocity of 3ft/sec.
(From Mellish, 1985, p43.)

may be usefully restricted by certain constraints arising from the rest of the sentence. For this sentence to make sense, the particle in question must be capable of moving. Similarly, it must not be known to be located at some point other than A; and so on. However, while these constraints should certainly be checked, they should not be allowed to actually refine the set of candidate particles in the manner of Mellish's program. For if there are two particles in the context, one fixed and the other mobile, the use of the definite article seems infelicitous.

Consider this point in the light of the following context. Suppose there are two toy trains on a toy train track. One is stationary and the other is moving around the track. In describing what happens next, a speaker cannot say:

(15) # The train starts to move.

Although, logically, only one of the trains can start to move, the use of the simple definite expression *the train* is inappropriate in (15). Depending on the prior linguistic context, the speaker would use some other expression, such as *the other train* or *the train that was stationary*. The presuppositions of the sentence in (16) are violated for a similar reason.

(16) # An electrical charge starts the train.

So it does not seem to matter whether the constraining information precedes or succeeds the definite NP: in either case, it should not be allowed to constrain the reference if the correct predictions are to be made.

Certain imperative constructions seem to exhibit the same behaviour. Consider an immediate physical situation involving a wash basin, in which one tap is running and one is not. A speaker would not simply say

(17) Turn on the tap

without somehow first bringing the non-running tap into focus, or accompanying (17) with an appropriate gesture; despite the fact that there is only one tap which, in the particular situation, can be turned on. (Note that the speaker probably *could* say *Turn off the tap*, but this is presumably because a running tap is generally more salient than a non-running tap.)

However, recall the rabbit scenario of Figure 1.1 and consider the following command, involving a simple NP *the rabbit* and a verbal argument *out of the hat*:

(18) Take the rabbit out of the hat

In (18), neither of the definite expressions *the rabbit* and *the hat* refer uniquely when considered in isolation, and yet the sentence as a whole appears to be acceptable. Pragmatic knowledge tells the hearer that (18) presupposes the rabbit in question is presently contained in a hat, and it is this knowledge which seems crucial in finding a unique referent for *the rabbit*. Likewise for the referent of *the hat*. So this piece of data suggests that Mellish is correct in claiming that definite references cannot be resolved locally, in contrast to the observations above.

So let us now consider some of the other examples which motivated Mellish's "lazy" approach to uniqueness. In the following example, the NP *the top* can only refer uniquely by considering constraints from the enclosing text:

(19) Into an open box is placed a wedge with half its horizontal dimensions. It extends 3 inches above *the top* and also holds an inelastic ball.

(From Mellish, 1985, p48)

Here Mellish's program cleverly deduces that *the top* must refer to the top of the wedge: the referent of *it* (in the second sentence) must be able to hold an inelastic ball and so *it* must refer to the box; but the box cannot extend above its own top, and so *the top* must be referring to the top of the wedge. However, cognitively, this example is rather difficult to process, and we might suspect that the use of *the top* in (19) is in fact infelicitous. But Mellish provides another, perhaps more natural, example:

(20) A hollow vertical cylinder, of radius 2a and height 3a, rests on a horizontal table, and a uniform rod is placed within it with its lower end resting on the circumference of *the base*.

(From Mellish, 1985, p20)

If *the base* refers naturally and without trouble to *the base of the cylinder*, then, as Mellish argues, it clearly must do so by virtue of the preceding material. It is not clear why *the base* refers successfully in (20), though it is worth noting that, as in (19), the type of reference involves a "bridging" inference (cf. Clark 1977) where *the base* refers back to the base implicitly introduced by *a hollow vertical cylinder*.

Now consider Ritchie's (1976) claim that a subsequent temporal quantifier may affect the referent of an earlier definite expression:

(21) a. *The President of the United States* signed the test-ban treaty *in 1962*
b. *The Tay Bridge* was blown down *in 1879*

(From Ritchie, 1976)

Recall (from Chapter 2) Ritchie's argument that the referent of *The President of the United*

States cannot be determined until the modifier *in 1962* is read, at which stage specific knowledge about political history can be used to pinpoint a particular individual. Similarly, there have been two Tay Bridges, but only the first one was blown down in 1879. Ritchie's argument stems from a strict Fregean distinction between sense and reference, but there is no need for such a rigid distinction between the two. In (21b), we *can* regard the *the Tay Bridge* as referring uniquely, in isolation of what follows, if we assume that it refers to some "token" entity in the hearer's mind representing *the thing that spans the River Tay*. Correspondingly, there is no need for *the President of the United States* in (21a) to refer to some specific individual known to the hearer for the sentence as a whole to be comprehensible. Thus, in the present terms, where we assume reference is to mental entities rather than real-world entities, it seems unreasonable to maintain that the expression *the President of the United States* does not refer uniquely in isolation.

This selection of data demonstrates that the precise processing point at which a definite NP can be expected to refer uniquely is difficult to pin down, as is the kind of semantic information which can or cannot help effect that unique reference. But the unquestionable infelicity of examples like (15), (16), and (12) in the context of (13), suggests that proposals to constrain a definite reference with pragmatic constraints arising from outside the NP should be approached with caution. The issue is an interesting one, and our discussion has perhaps shown that even the simple kinds of NP considered by this thesis deserve closer linguistic and psycholinguistic investigation.

7.2. Areas for Further Work

This final section considers some possibly fruitful ways of extending the model of incremental interpretation to cover a wider range of linguistic applications.

For example, it would clearly be helpful to have an account of new as well as given information. As we saw in section 7.1, this would enable us to look at examples at the sentence level and consider how main clause processing interacts with reference resolution. However, getting a sentence processor to make up its own mind about the informational status of each fragment of the linguistic input is difficult. Mellish's (1985) program makes the decision on purely syntactic grounds, assuming that all material conveys new information, unless it is within the bounds of a definite noun phrase, in which case it is assumed to be given. Although this may be adequate for the domain in which his program operates, it is generally acknowledged that informational status also depends on contextual factors, such as the local structure of the discourse and the particular states of mind of speaker and

hearer.

Hobbs, Stickel, Martin and Edwards (1988) adopt a more pragmatic position in the TACITUS project (see also Hobbs, 1986; Hobbs and Martin, 1987; Bear and Hobbs, 1988). Broadly speaking, their approach is to first try to prove that a piece of information follows from existing facts in the system's knowledge-base, and if such a proof can be made assume that the information is given. On the other hand, if there is no chain of inference to connect the predicates associated with the linguistic input to the propositions in the knowledge-base, the information is assumed to be new.

Allowing definite noun phrases to introduce entities into the context would, among other things, provide a basis for a computational correlate of Crain and Steedman's Principle of Parsimony (cf. Chapter 2), which states that the HSPM will favour the reading which makes fewest modifications to the context. Indeed, the TACITUS system is very relevant here, for it operates under what Hobbs (1986, p221) calls the "Minimality Principle": it favours the solution which hypothesises the "fewest new entities and relations." This is achieved by assigning a cost to each addition to the knowledge-base, and then adopting a general strategy of favouring the least expensive inference paths. As the Principle of Parsimony suggests, in the case of a syntactically ambiguous sentence, Hobbs' processing regime will favour the reading which makes fewest additions to the context.

In fact, it is quite apparent that much of Hobbs' research programme is of direct relevance to the present endeavour. The reason for this is simple: both projects aim for an integration of syntactic, semantic and referential processing, and both emphasise the importance of reference to the context as a primary means of syntactic and semantic disambiguation. Of course, the TACITUS project confronts numerous issues which we have not even mentioned here. And the two projects differ in almost all technical details. But, in spirit at least, the two projects have much in common, and Hobbs' work would be a sensible place to look for ways of extending the program presented here.

By allowing sentences to add information to the current context as well as extract information from it, we would also pave the way for a more general semantic treatment of noun phrases and other referring expressions. Indefinites could optionally introduce entities, as they commonly do, and noun modifiers could be optionally interpreted as non-restrictive (i.e. as providing new information) instead of just being restrictive. As for other syntactic types of noun phrase, Mellish (1985) discusses a wide variety of plural and quantified noun phrases, and so his work obviously provides a sensible starting point for anyone wishing to generalise the work here. The unimplemented ideas of Bobrow and Webber

(1980a) and the more detailed proposals of Hobbs (1983) are also very relevant, since both emphasise the need for vague representations of the ambiguities that can arise from plural forms, and we have seen how such vague descriptions are useful for incremental processing.

How should one go about extending the present model to encompass these other varieties of reference? It would be natural to start with the plural and quantified NPs, since they are commonly viewed in terms of *sets*, and such referential sets are already taken as a primitive of the model. At present, we associate each semantic variable with a domain of values. For a singular reference, we assume that the value of this variable is one or other of the elements in its domain. One approach to plural reference might be to introduce a semantic variable whose value is the domain itself. Thus we should be able to treat plural reference by elaborating the semantic translations and the way in which they are interpreted, with respect to the same underlying process of network consistency. Meanwhile, the problem of quantifier scoping might be approached from the perspective of closure constraints. We could associate each quantifier with an appropriate closure constraint, and capture the different scope interpretations by allowing the closure constraints to be discharged from the closure list in a number of possible sequences (possibly along the lines of Pollack and Pereira's (1988) recent work). So here we would address the quantifier scope problem solely in terms of manipulations of the closure list, interpreting the different scopings with respect to the same underlying logical form and extensions.

However, other forms of reference would require more significant extensions. For example, allowing indefinite NPs to introduce new entities to the context would require an evaluation procedure which goes beyond the network consistency process used for given expressions. Similarly, other uses of definite NPs, such as the attributive use, might not be best seen in the purely extensional framework offered here.

Pronouns are a topic of particularly intense interest in the literature, and have been studied in a wide variety of semantic frameworks. Of these, Kamp's (1981) Discourse Representation Theory might be helpful in extending the present work, given that there is at least a superficial similarity between aspects of his discourse representations and our CSP-oriented semantic translations. (See also Webber (1978) and Heim (1982) for frameworks similar to Kamp's.) The work of Johnson and Klein (1986) and Pollack and Pereira (1988) is especially relevant, since it discusses anaphora resolution as an operation on-line to the overseeing sentence processor.

It seems likely that the timing of the resolution process will need careful consideration in the pronoun case, for, unlike full definite noun phrases, we cannot expect the reference of a pronoun to be resolved locally, as soon as it is read (modulo the discussion of definite reference in section 7.1). Consider the following well-known pair of sentences, for example:

(22) a. The councillors refused the demonstrators a permit because they feared violence
b. The councillors refused the demonstrators a permit because they advocated revolution

Here, common-sense knowledge tells us that in this situation the councillors are more likely to fear violence than the demonstrators, and that the demonstrators are more likely to be advocating revolution than the councillors. In the absence of a more specific, referential context, we appear to use this knowledge to resolve the reference of *they* in (a) to the councillors, and in (b) to the demonstrators. But notice that in both cases the disambiguating material comes after the pronoun, not before it. So, as Mellish indeed argues, we cannot necessarily expect a pronominal NP to refer successfully by the time it is syntactically closed.

Finally, although geared towards theoretical concerns, the model of syntax and semantics discussed here may have a practical role to play in systems which encounter syntactically incomplete sentences among their linguistic input. Combinatory grammar gives syntactic status to a range of sentence fragments, and the semantics we associated with it in Chapter 5 allows each of these fragments to be evaluated in the context. Hinrichs and Polanyi (1986) have already demonstrated the usefulness of CCG in assigning a syntactic category to partial sentences like *We could go to*, where the missing argument is provided by a referential gesture rather than by a noun phrase.

Kobsa *et al* (1986) describe a front-end to an expert system which similarly mixes graphical with natural language input. Such systems need to integrate the processing of deictic, gestural input with partial linguistic input, since both sources of information may be simultaneously required to determine the particular referent intended by the speaker. Within the present framework, linguistic and extra-linguistic information alike could translate into network constraints on the set of available referents. And it may even be useful to formulate the referent identification process as a left-to-right, incremental procedure, as Kobsa *et al* have indeed done (Allgayer, personal communication).

References

Ades, A. and Steedman, M. J. (1982) On the Order of Words. *Linguistics and Philosophy*, **4**, 517-558.

Aho, A. V. and Ullman, J. D. (1972) *The Theory of Parsing, Translation, and Compiling*, Volume 1: *Parsing*. New Jersey: Englewood Cliffs.

Ajdukiewicz, K. (1935) Die syntaktische Konnexitat. *Studia Philosophica*, **1**, 1-27. English translation in: *Polish Logic: 1920-1939* , ed. by Storrs McCall, pp207-231, Oxford University Press, Oxford.

Allen, J. F. (1987) *Natural Language Understanding*. Menlo Park, Ca.: The Benjamin/Cummings Publishing Co.

Altmann, G. T. M. (1985) The Resolution of Local Syntactic Ambiguity by the Human Sentence Processing Mechanism. In *Proceedings of the 2nd European Meeting of the Association for Computational Linguistics*, Geneva, pp123-127.

Altmann, G. T. M. (1986) Reference and the Resolution of Local Ambiguity: Interaction in Human Sentence Processing. PhD Thesis, University of Edinburgh.

Altmann, G. T. M. (1987) Modularity and Interaction in Sentence Processing. In Garfield, J. (ed.) *Modularity in Knowledge Representation and Natural Language Processing*, pp428-444. Cambridge, Mass.: MIT Press.

Altmann, G. T. M. (1988) Ambiguity, Parsing Strategies, and Computational Models. *Language and Cognitive Processes*, **3**.

Altmann, G. T. M. and Steedman, M. J. (1988) Interaction with Context during Human Sentence Processing. *Cognition*, **30**.

Barton, G. E. (1986) Constraint Propagation in Kimmo Systems. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, N.Y., pp45-52.

Barton, G. E., Berwick, R. C. and Ristad, E. S. (1987) *Computatational Complexity and Natural Language*. Cambridge, Mass.: MIT Press.

Bates, M., Bobrow, R. and Webber, B. (1981) Tools for Syntactic and Semantic Interpretation. Annual Report No. 4785, Bolt, Beranek and Newman Inc., Cambridge, Mass..

Bear, J. (1983) A Breadth-First Parsing Model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp696-698.

Bear, J. and Hobbs, J. R. (1988) Localizing Expression of Ambiguity. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas, pp235-242.

Berwick, R. C. and Weinberg, A. (1984) *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition*. Cambridge, Mass.: MIT Press.

Bever, T. G. (1970) The Cognitive Basis for Linguistic Structures. In Hayes, J. R. (ed.) *Cognition and the Development of Language*. New York: John Wiley and Sons.

Bobrow, R. J. and Webber, B. L. (1980a) PSI-KLONE: Parsing and Semantic Interpretation in the BBN Natural language understanding system. In *Proceedings of the Canadian Society for Computational Studies of Intelligence*, pp131-142.

Bobrow, R. J. and Webber, B. L. (1980b) Knowledge Representation for Syntactic/Semantic Processing. In *Proceedings of the First Annual National Conference on Artificial Intelligence*, Stanford, Ca..

Bollobas, B. (1979) *Graph Theory: An Introductory Course*. Berlin: Springer-Verlag.

Brachman, R. J. (1979) Taxonomy, Descriptions, and Individuals in Natural Language Understanding. In *Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics*, University of California at San Diego, La Jolla, Ca., pp33-38.

Brachman, R. J. and Schmolze, J. G. (1985) An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, **9**, 171-216.

Bundy, A., Byrd, L., Luger, G., Mellish, C. and Palmer, M. (1979) Solving Mechanics Problems using Meta-Level Inference. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan.

Burton, R. R. (1976) Semantic Grammar: An Engineering technique for Constructing Natural Language Systems. BBN Report No. 3453, Bolt, Beranek and Newman Inc., Cambridge, Mass..

Calder, J., Klein, E., Moens, M. and Zeevat, H. (1986) Problems of Dialogue Parsing. ACORD Deliverable T2.1, Centre for Cognitive Science, Edinburgh.

Calder, J., Klein, E. and Zeevat, H. (1988) Unification Categorial Grammar: A Concise, Extendable Grammar for Natural Language Processing. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, pp83-86.

Charniak, E. (1986) A Neat Theory of Marker Passing. In *Proceedings of the 5th Annual Meeting of the American Association for Artificial Intelligence*, Philadelphia, Pa., pp584-588.

Charniak, E. and Goldman, R. (1988) A Logic for Semantic Interpretation. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, State University of New York at Buffalo, Buffalo, N.Y., pp87-94.

Clark, H. H. and Haviland, S. E. (1977) Comprehension and the Given-New Contract. In Freedle, R. O. (ed.) *Discourse Production and Comprehension*, Volume 1, pp1-40. Norwood, N.J.: Ablex.

Clark, H. H. (1977) Bridging. In Johnson-Laird, P. N. and Wason, P. C. (eds.) *Thinking: Readings in Cognitive Science*, pp411-420. Cambridge: Cambridge University Press.

Clark, H. H. and Marshall, C. R. (1981) Definite Reference and Mutual Knowledge. In Joshi, A. K., Webber, B. L. and Sag, I. A. (eds.) *Elements of Discourse Understanding*, pp10-63. Cambridge: Cambridge University Press.

Crain, S. (1980) Pragmatic Constraints on Sentence Comprehension. Unpublished PhD dissertation, University of California at Irvine.

Crain, S. and Steedman, M. J. (1985) On Not Being Led up the Garden Path: The Use of Context by the Psychological Parser. In Dowty, D., Karttunen, L. and Zwicky, A. (eds.) *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*.

Dahlgren, K. and McDowell, J. (1986) Using Commonsense Knowledge to Disambiguate Prepositional Phrase Modifiers. In *Proceedings of the 5th Annual Meeting of the American Association for Artificial Intelligence*, Philadelphia, Pa., pp589-593.

Davidson, D. (1967) The Logical Form of Action Sentences. In Rescher, N. (ed.) *The Logic of Decision and Action*. Pittsburgh: University of Pittsburgh Press.

Davis, E. (1987) Constraint Propagation with Interval Labels. *Artificial Intelligence*, **32**, 281-331.

Donnellan, K. (1966) Reference and Definite Descriptions. *Philosophical Review*, **75**, 281-304.

Duffy, G. (1986) Categorial Disambiguation. In *Proceedings of the 5th Annual Meeting of the American Association for Artificial Intelligence*, Philadelphia, Pa., pp1079-1082.

Engdahl, E. (1983) Parasitic Gaps. *Linguistics and Philosophy*, **6**, 5-34.

Fenstad, J. E., Halvorsen, P., Langholm, T. and Benthem, J. (1985) Equations, Schemata and Situations: a Framework for Linguistic Semantics. Technical Report No. CSLI-85-29.

Ferreira, F. and Clifton, C. (1986) The Independence of Syntactic Processing. *Journal of Memory and Language*, **25**, 348-368.

Fodor, J. D. and Frazier, L. (1980) Is the Human Sentence Parsing Mechanism an ATN? *Cognition*, **8**, 418-459.

Ford, M., Bresnan, J. and Kaplan, R. M. (1982) A Competence-based Theory of Syntactic Closure. In Bresnan, J. (ed.) *The Mental Representation of Grammatical Relations*. Cambridge, Mass.: MIT Press.

Frazier, L. (1978) On Comprehending Sentences: Syntactic Parsing Strategies. PhD Thesis, University of Connecticut. Indiana University Linguistics Club.

Frazier, L. and Fodor, J. D. (1978) The Sausage Machine: a new two-stage parsing model. *Cognition*, **6**, 291-325.

Freuder, E. C. (1978) Synthesizing Constraint Expressions. *Communications of the ACM*, **21**, 958-966.

Freuder, E. C. (1982) A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, **19**, 24-32.

Freuder, E. C. (1985) A Sufficient Condition for Backtrack-Bounded Search. *Journal of the ACM*, **32**.

Garey, M. R. and Johnson, D. S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.

Garrod, S. C. and Sanford, A. J. (1985) On the Real-Time Character of Interpretation during Reading. *Language and Cognitive Processes*, **1**, 43-59.

Gawron, J. M., King, J., Lamping, J., Loebner, E., Paulson, A., Pullum, G. K., Sag, I. A. and Wasow, T. (1982) The GPSG Linguistics System. In *Proceedings of the 20st Annual Meeting of the Association for Computational Linguistics*, University of Toronto, Toronto, Ontario, Canada, pp74-81.

Gehrke, M. (1983) Syntax, Semantics and Pragmatics in Concert: An Incremental, Multi-Level Approach in Reconstructing Task-Oriented Dialogues. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp721-723.

Goodman, B. A. (1986) Reference Identification and Reference Identification Failures. *Computational Linguistics*, **12**, 273-305.

Grice, H. P. (1975) Logic and Conversation. In Cole, P. and Morgan, J. L. (eds.) *Syntax and Semantics*, Volume 3: *Speech Acts*, pp41-58. New York: Academic Press.

Grosz, B. J. (1977) The Representation and Use of Focus in Dialogue. Technical Note No. 151, SRI International, Menlo Park, Ca..

Haddock, N. J. (1987) Incremental Interpretation and Combinatory Categorial Grammar. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, pp661-663.

Halliday, M. A. K. (1967) Notes on Transitivity and Theme in English: Part Two. *Journal of Linguistics*, **3**, 199-244.

Halvorsen, P. (1986) Natural Language Understanding and Montague Grammar. *Computational Intelligence*, **2**, 54-62.

Halvorsen, P. (1987) Situation Semantics and Semantic Interpretation in Constraint-based Grammars. Report No. CSLI-87-101, Center for the Study of Language and Information.

Hawkins, J. A. (1978) *Definiteness and Indefiniteness*. London: Croom Helm.

Heim, I. (1982) The Semantics of Definite and Indefinite Noun Phrases. PhD Thesis, University of Massachusetts. Distributed by Graduate Linguistic Student Association.

Hepple, M. (1987) Methods for Parsing Combinatory Grammars and the Spurious Ambiguity Problem. Masters Thesis, Centre for Cognitive Science, University of Edinburgh.

Hinrichs, E. and Polanyi, L. (1986) Pointing the Way. In *Papers from the Parasession on Pragmatics and Grammatical Theory at the Twenty-Second Regional Meeting of the Chicago Linguistic Society*, Chicago, pp298-314.

Hirst, G. (1983a) A Foundation for Semantic Interpretation. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Massachusetts Institute of Technology, Cambridge, Mass., pp64-73.

Hirst, G. (1983b) Semantic Interpretation against Ambiguity. Technical Report No. CS-83-25, Dept. of Computer Science, Brown University.

Hirst, G. (1984) A Semantic Process for Syntactic Disambiguation. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, Austin, Texas, pp148-152.

Hirst, G. (1987) *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge: Cambridge University Press.

Hirst, G. (1988) Semantic Interpretation and Ambiguity. *Artificial Intelligence*, **34**, 131-177.

Hobbs, J. R. (1983) An Improper Treatment of Quantification in Ordinary English. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Massachusetts Institute of Technology, Cambridge, Mass., pp57-63.

Hobbs, J. R. (1985) Ontological Promiscuity. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, University of Chicago, Chicago, Illinois, pp61-69.

Hobbs, J. R. (1986) Overview of the TACITUS Project. *Computational Linguistics*, **12**.

Hobbs, J. R. and Martin, P. (1987) Local Pragmatics. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, pp520-523.

Hobbs, J. R., Stickel, M., Martin, P. and Edwards, D. (1988) Interpretation as Abduction. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, State University of New York at Buffalo, Buffalo, N.Y., pp95-103.

Isard, S. D. (1975) Changing the Context. In Keenan, E. (ed.) *Formal Semantics of Natural Language*. Cambridge: Cambridge University Press.

Jacobs, P. S. (1987) A Knowledge Framework for Natural Language Analysis. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, pp675-678.

Johnson, M. and Klein, E. (1986) Discourse, Anaphora and Parsing. In *Proceedings of the 11th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics*, Institut fuer Kommunikationsforschung und Phonetik, Bonn University, Bonn, pp669-675.

Kamp, H. (1981) A Theory of Truth and Semantic Representation. In Groenendijk, J. A. G., Janssen, T. M. V. and Stokhof, M. B. J. (eds.) *Formal Methods in the Study of Language*, Volume 136, pp277-322. Amsterdam: Mathematical Centre Tracts.

Kartunnen, L. (1986) D-PATR: A Development Environment for Unification-Based Grammars. In *Proceedings of the 11th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics*, Institut fuer Kommunikationsforschung und Phonetik, Bonn University, Bonn, pp74-80.

Kay, M. (1980) Algorithm Schemata and Data Structures in Syntactic Processing. Technical Report No. CSL-80-12, XEROX Palo Alto Research Centre, Palo Alto.

Kimball, J. (1973) Seven Principles of Surface Structure Parsing in Natural Language. *Cognition*, **2**, 15-47.

Kobsa, A., Allgayer, J., Reddig, C., Reithinger, N., Schmauks, D., Harbusch, K. and Wahlster, W. (1986) Combining Deictic Gestures and Natural Language for Referent Identification. In *Proceedings of the 11th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics*, Institut fuer Kommunikationsforschung und Phonetik, Bonn University, Bonn, pp356-361.

Lesmo, L. and Torasso, P. (1985) Weighted Interaction of Syntax and Semantics in Natural Language Analysis. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, University of California at Los Angeles, Los Angeles, Ca., pp772-778.

Lytinen, S. L. (1984) The Organization of Knowledge in a Multi-lingual, Integrated Parser. PhD Thesis, Computer Science, Yale University.

Mackworth, A. K. (1977a) Consistency in Networks of Relations. *Artificial Intelligence*, **8**, 99-118.

Mackworth, A. K. (1977b) On Reading Sketch Maps. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, MIT, Cambridge, Mass., pp598-606.

Mackworth, A. K. and Freuder, E. C. (1985) The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, **25**, 65-74.

Mackworth, A. K. (1987) Constraint Satisfaction. In Shapiro, S. (ed.) *The Encyclopedia of Artificial Intelligence*, Volume 1, pp205-211. John Wiley and Sons.

Marcus, M. P. (1980) *A Theory of Syntactic Recognition for Natural Language*. Cambridge, Mass.: MIT Press.

Marslen-Wilson, W. D. (1975) Sentence Perception as an Interactive Parallel Process. *Science*, **189**, 226-228.

Marslen-Wilson, W. and Tyler, L. K. (1980) The Temporal Structure of Spoken Language Understanding. *Cognition*, **8**, 1-74.

Marslen-Wilson, W. D. (1987) Functional Parallelism in Spoken Word Recognition. *Cognition*, **25**, 71-102.

McCawley, J. D. (1968) The Role of Semantics in a Grammar. In Bach, E. and Harms, R. T. (eds.) *Universals in Linguistic Theory*, pp125-169. New York: Holt, Rinehart and Winston.

Mellish, C. S. (1981) Coping with Uncertainty: Noun Phrase Interpretation and Early Semantic Analysis. PhD Thesis, University of Edinburgh.

Mellish, C. (1982) Incremental Evaluation: An Approach to the Semantic Interpretation of Noun Phrases. Cognitive Science Research Paper No. 1, Cognitive Studies Programme, University of Sussex, Brighton.

Mellish, C. S. (1983) Incremental Semantic Interpretation. In Sparck-Jones and Wilks (eds.) *Parsing Natural Language*.

Mellish, C. S. (1985) *Computer Interpretation of Natural Language Descriptions*. Chichester: Ellis Horwood.

Milne, R. W. (1982) Predicting Garden Path Sentences. *Cognitive Science*, **6**, 349-373.

Montague, R. (1973) The Proper Treatment of Quantification in Ordinary English. In Hintikka, J., Moravcsik, J. M. E. and Suppes, P. (eds.) *Approaches to Natural Language*. Dordrecht: D. Reidel.

Montanari, U. (1974) Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, **7**, 95-132.

Moortgat, M. (1988) *Categorial Investigations*: *Logical and Linguistic Aspects of the Lambek Calculus*. Dordrecht: Foris Publications.

Morrill, G. (1988) Extraction and Coordination in Phrase Structure Grammar and Categorial Grammar. PhD Thesis, Centre for Cognitive Science, University of Edinburgh.

Palmer, M. S., Dahl, D. A., Schiffman, R. J., Hirschman, L., Linebarger, M. and Dowding, J. (1986) Recovering Implicit Information. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, N.Y., pp10-19.

Pareschi, R. and Steedman, M. J. (1987) A Lazy Way to Chart-Parse with Extended Categorial Grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, Ca..

Partee, B. (1978) Bound Variables and Other Anaphors. In Waltz, D. L. (ed.) *Theoretical Issues in Natural Language Processing-2*, University of Illinois at Urbana-Champaign, Urbana, Illinois, pp79-85.

Pazzani, M. J. (1984) Conceptual Analysis of Garden-Path Sentences. In *Proceedings of the 10th International Conference on Computational Linguistics and the 22nd Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, Ca., pp486-490.

Pereira, F. C. N. and Warren, D. H. D. (1980) Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Grammars. *Artificial Intelligence*, **13**, 231-278.

Pereira, F. C. N. (1985) A New Characterization of Attachment Preferences. In Dowty, D., Karttunen, L. and Zwicky, A. (eds.) *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, pp307-319. Cambridge: Cambridge University Press.

Pereira, F. C. N. and Shieber, S. M. (1987) *Prolog and Natural Language Analysis*. Center for the Study of Language and Information. CSLI Lecture Notes No. 10.

Pollack, M. E. and Pereira, F. C. (1988) An Integrated Framework for Semantic and Pragmatic Interpretation. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, State University of New York at Buffalo, Buffalo, N.Y., pp75-86.

Pulman, S. G. (1986) Grammars, Parsers, and Memory Limitations. *Language and Cognitive Processes*, **1**, 197-225.

Rayner, K., Carlson, M. and Frazier, L. (1983) The Interaction of Syntax and Semantics during Sentence Processing. *Journal of Verbal Learning and Verbal Behavior*, **22**, 358-374.

Reinhart, T. (1983) Coreference and Bound Anaphora: A restatement of the anaphora questions. *Linguistics and Philosophy*, **6**, 47-88.

Rich, E. (1983) *Artificial Intelligence*. New York: McGraw Hill.

Rich, E., Wittenburg, K., Barnett, J. and Wroblewski, D. (1987) Ambiguity Procrastination. In *Proceedings of the 6th Annual Meeting of the American Association for Artificial Intelligence*, Seattle, pp571-576.

Riesbeck, C. K. (1975) Computational Understanding. In Schank, R. and Nash-Webber, B. L. (eds.) *Theoretical Issues in Natural Language Processing*, Cambridge, Mass.

Ritchie, G. D. (1976) Problems in Local Semantic Processing. In Brady, M. (ed.) *Proceedings of the AISB Conference*, University of Edinburgh.

Ritchie, G. D. (1983) Semantics in Parsing. In King, M. (ed.) *Natural Language Parsing*.

Rosenschein, S. J. and Shieber, S. M. (1982) Translating English into Logical Form. In *Proceedings of the 20st Annual Meeting of the Association for Computational Linguistics*, University of Toronto, Toronto, Ontario, Canada, pp1-8.

Ross, J. R. (1967) Constraints on Variables in Syntax. PhD Thesis, MIT. Indiana University Linguistics Club.

Schubert, L. and Pelletier, J. (1982) From English to Logic: Context-Free Computation of 'Conventional' Logical Translation. *American Journal of Computational Linguistics*, **8**, 27-44.

Schubert, L. K. (1984) On Parsing Preferences. In *Proceedings of the 10th International Conference on Computational Linguistics and the 22nd Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, Ca..

Schubert, L. K. (1986) Are there Preference Trade-offs in Attachment Decisions?. In *Proceedings of the 5th Annual Meeting of the American Association for Artificial Intelligence*, Philadelphia, Pa., pp601-605.

Shieber, S. (1983) Sentence Disambiguation by a Shift-Reduce Parsing Technique. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Massachusetts Institute of Technology, Cambridge, Mass., pp113-118.

Shillcock, R. (1982) The On-line Resolution of Pronominal Anaphora. *Language and Speech*, **25**, 385-402.

Sidner, C. L. (1979) Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse. Technical Report No. 537, MIT Artificial Intelligence Laboratory.

Sondheimer, N. K., Weischedel, R. M. and Bobrow, R. J. (1984) Semantic Interpretation using KL-ONE. In *Proceedings of the 10th International Conference on Computational Linguistics and the 22nd Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, Ca., pp101-107.

Steedman, M. (1985) Dependency and Coordination in the Grammar of Dutch and English. *Language*, **61**, 523-568.

Steedman, M. J. (1987a) Combinatory Grammars and Human Language Processing. In Garfield, J. (ed.) *Modularity in Knowledge Representation and Natural Language Processing*. Cambridge, Mass.: Bradford/MIT Press.

Steedman, M. (1987b) Combinatory Grammars and Parasitic Gaps. *Natural Language and Linguistic Theory*, **5**, 403-439.

Steedman, M. (1988) Combinators and Grammars. In Oehrle, R., Bach, E. and Wheeler, D. (eds.) *Categorial Grammars and Natural Language Structures*, Dordrecht. Paper presented at the Conference on Categorial Grammar, Tucson, June 1985.

Steedman, M. J. (in press) Constituency and Coordination in a Combinatory Grammar. In Baltin, M. (ed.) *Alternative Conceptions of Phrase Structure*. Chicago, IL: University of Chicago Press. (in press).

Steedman, M. J. (forthcoming) Syntax and Intonational Structure in a Combinatory Grammar. In Altmann, G. T. M. (ed.) *Cognitive Models of Speech Processing: Psycholinguistic and Computational Perspectives*. Cambridge, Mass.: MIT Press.

Swinney, D. A. (1979) Lexical Access during Sentence Comprehension: (re)consideration of context effects. *Journal of Verbal Learning and Verbal Behavior*, **18**, 645-660.

Tanenhaus, M. K., Carlson, G. N. and Seidenberg, M. S. (1985) Do Listeners Compute Linguistic Representations? In Dowty, D. R., Karttunen, L. and Zwicky, A. M. (eds.) *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pp359-408. Cambridge: Cambridge University Press.

Tomabechi, H. (1987) Direct Memory Access Translation. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, pp722-725.

Tomita, M. (1987) An Efficient Augmented-Context-Free Parsing Algorithm. *Computational Linguistics*, **13**, 31-46.

Tomita, M. (1988) Graph-Structured Stack and Natural Language Parsing. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, State University of New York at Buffalo, Buffalo, N.Y., pp249-257.

Trehan, R. and Wilk, P. (1987) A Parallel Shift-Reduce Parser for Committed Choice Non-Deterministic Logic Languages. Research Paper No. 327, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland.

Tyler, L. and Marslen-Wilson, W. D. (1977) The On-line Effects of Semantic Context on Syntactic Processing. *Journal of Verbal Learning and Verbal Behavior*, **16**, 683-692.

Waltz, D. (1975) Understanding Line Drawings of Scenes with Shadows. In Winston, P. H. (ed.) *The Psychology of Computer Vision*, pp19-91. New York: McGraw Hill.

Waltz, D. and Pollack, J. (1985) Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. *Cognitive Science*, **9**, 51-74.

Wanner, E. (1980) The ATN and the Sausage Machine: which one is baloney? *Cognition*, **8**, 209-225.

Wanner, E. (1987) The Parser's Architecture. In *The Development of Language and Language Researchers: Essays in honour of Roger Brown*. Erlbaum.

Webber, B. L. (1978) A Formal Approach to Discourse Anaphora. PhD Thesis, Harvard University.

Weischedel, R. M. (1979) A New Semantic Computation while Parsing: Presupposition and Entailment. In Oh, C. and Dineen, D. (eds.) *Syntax and Semantic II: Presupposition*, pp155-182. New York: Academic Press.

Wilks, Y., Huang, X. and Fass, D. (1985) Syntax, Preference and Right Attachment. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp779-84.

Winograd, T. (1971) Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Technical Report No. MAC TR-84, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Winograd, T. (1972) *Understanding Natural Language*. Edinburgh: Edinburgh University Press.

Winograd, T. (1973) A Procedural Model of Language Understanding. In Schank, R. C. and Colby, K. (eds.) *Computer Models of Thought and Language*. San Fransisco: W. H. Freeman.

Winston, P. H. (1984) *Artificial Intelligence*. Reading, Mass.: Addison-Wesley.

Wittenburg, K. W. (1986) Natural Language Parsing with Combinatory Categorial Grammar in a Graph-Unification-Based Formalism. PhD Thesis, Department of Linguistics, University of Texas.

Wittenburg, K. (1987) Predictive Combinators: a Method for Efficient Processing of Combinatory Categorial Grammar. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, Ca..

Woods, W. A. (1970) Transition Network Grammars for Natural Language Analysis. *Communications of the ACM*, **13**, 591-606.

Woods, W. A., Kaplan, R. M. and Nash-Webber, B. L. (1972) The Lunar Sciences Natural Language Information System: Final Report. BBN Report No. 2378, Bolt, Beranek and Newman Inc., Cambridge, Mass..

Woods, W. (1973) An Experimental Parsing System for Transition Network Grammars. In Rustin, R. (ed.) *Natural Language Processing*, pp111-154. New York: Algorithmics Press.

Woods, W. (1980) Cascaded ATN Grammars. *American Journal of Computational Linguistics*, **6**.

Appendix A

A Prolog Implementation

The algorithms described in Chapters 5 and 6 have been implemented in C Prolog under Unix on a VAX 11/750. The system's grammar includes the rules of forward and backward application, forward composition and crossed backward composition. A small lexicon has been written to deal with the NPs addressed in Chapter 5, and so includes determiners, nouns, adjectives, verbs, relative pronouns, and noun- and verb-modifying prepositions.

The reference evaluator is implemented as described, incorporating Mackworth's network consistency algorithm and the closure constraint mechanism. However, the more general scheme of closure constraints based on numerical closure points (see section 5.3.4) has not yet been coded, and for this reason generalised forward composition is not currently included in the program's set of combinatory rules.

The incremental reference evaluation procedures are at present driven by the breadth-first chart parsing algorithm of section 6.4, augmented with the simple check for redundancy described there and attributed to Karttunen. Given recent developments in this area, it would be appropriate to replace this redundancy check with Hepple and Morrill's more sophisticated equivalence relations.

For purpose of clarifying any ambiguities in the main text, the following sections provide the Prolog code for the main components of the system.

• A.1. Operator Declarations

The notation used in the program is similar to that used in the main text, except that a category written as

NP_{e_1}/N_{e_1}

in the main text is written as

np:E1/n:E1

in the program. So syntactic categories are seen as Prolog constants and semantic variables are seen as Prolog variables, and they are separated within categories by a colon rather than by subscripting. Given that colons now occur within categories, a

double colon "::" appears between categories and constraint lists in the lexicon, instead of the single colon used in the text. Note also that category slashes have been defined so that they associate to the *left*; X/Y/Z is taken to mean (X/Y)/Z, and so on.

- **A.2. Grammar and Lexicon**

A combinatory rule is represented as a four-place predicate

```
synRule(Type,C1,C2,C)
```

where each rule has Type = a (for application) or Type = b (for composition), where C1 and C2 are left-hand and right-hand categories, and C is their reduction. The Type field is used by the parser so that it can enforce closure constraints whenever a rule of application is used.

Since all combinatory rules concatenate closure constraints and network constraints, the semantic operation of concatenation is stated once and applies to all combinatory rules. In the following rule,

```
semRule(_,U1::T1,U2::T2,U::T):-
        append(U1,U2,U),
        append(T1,T2,T).
```

Type can be anything, U1, U2, and U are closure lists, and T1, T2, and T are network constraints.

- **A.3. Contexts**

The sets of contextual facts found in the text are implemented as Prolog database entries with the distinguishing prefix "#". Two example contexts are included, to accompany the lexicon in A.2. Each context file also creates an initial value domain consisting of all contextual entities in the file, for use by the consistency procedure.

- **A.4. Parser**

Low-level routines have been omitted from this and the following section.

- **A.5. Evaluation Procedures**

- **A.6. Test Runs**

The final section presents some examples of the output produced by the program as it parses and interprets a noun phrase.

```
%%      A.1.  Operator declarations


%       ':=' Separates words from definitions in the lexicon.
%       '::' Separates categories from constraints in the lexicon.
%       '::' Also separates closure constraints from network
%            constraints in the semantic translation rule.
%       '*'  Distinguishes an item in a lexical constraint list as
%            a closure constraint.
%       '/'  Forward slash within categories.
%       '\'  Backward slash within categories.
%       ':'  Separates an atomic syntactic category from its
%            corresponding semantic variable within categories.
%       '#'  Prefix to any context predicate.

:- op(650,xfx,':='),
   op(450,xfx,'::'),
   op(400,yfx,/),
   op(400,yfx,\),
   op(400,fx,#),
   op(400,fx,*),
   op(380,yfx,:).
```

```
%%      A.2.  Grammar and Lexicon


%%      Rules

%  Four syntactic rules:
%       forward, backward application
%       forward, crossed backward composition
%  Plus a semantic rule to concatenate constraints.


synRule(a, X/Y, Y, X).
synRule(a, Y, X\Y, X).
synRule(c, X/Y, Y/Z, X/Z).
synRule(c, Y/Z, X\Y, X/Z).


semRule(_,U1::T1,U2::T2,U::T):-
        append(U1,U2,U),
        append(T1,T2,T).



%%      Lexicon

%  Determiners

a       :=  np:E1/n:E1                          ::  [].
a       :=  s:E1/(s:E1\np:E2)/n:E2              ::  [].

the     :=  np:E1/n:E1                          ::  [* unique(E1)].
the     :=  s:E1/(s:E1\np:E2)/n:E2              ::  [* unique(E2)].


%  Nouns

woman   :=  n:E1                                ::  [woman(E1)].
woman   :=  n:E1/(n:E1\n:E1)                    ::  [woman(E1)].
woman   :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)        ::  [woman(E1)].

man     :=  n:E1                                ::  [man(E1)].
man     :=  n:E1/(n:E1\n:E1)                    ::  [man(E1)].
man     :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)        ::  [man(E1)].

girl    :=  n:E1                                ::  [girl(E1)].
girl    :=  n:E1/(n:E1\n:E1)                    ::  [girl(E1)].
girl    :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)        ::  [girl(E1)].

boy     :=  n:E1                                ::  [boy(E1)].
boy     :=  n:E1/(n:E1\n:E1)                    ::  [boy(E1)].
boy     :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)        ::  [boy(E1)].
```

```
town      :=  n:E1                                  ::  [town(E1)].
town      :=  n:E1/(n:E1\n:E1)                      ::  [town(E1)].
town      :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [town(E1)].

river     :=  n:E1                                  ::  [river(E1)].
river     :=  n:E1/(n:E1\n:E1)                      ::  [river(E1)].
river     :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [river(E1)].

rabbit    :=  n:E1                                  ::  [rabbit(E1)].
rabbit    :=  n:E1/(n:E1\n:E1)                      ::  [rabbit(E1)].
rabbit    :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [rabbit(E1)].

hat       :=  n:E1                                  ::  [hat(E1)].
hat       :=  n:E1/(n:E1\n:E1)                      ::  [hat(E1)].
hat       :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [hat(E1)].

block     :=  n:E1                                  ::  [block(E1)].
block     :=  n:E1/(n:E1\n:E1)                      ::  [block(E1)].
block     :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [block(E1)].

box       :=  n:E1                                  ::  [box(E1)].
box       :=  n:E1/(n:E1\n:E1)                      ::  [box(E1)].
box       :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [box(E1)].

table     :=  n:E1                                  ::  [table(E1)].
table     :=  n:E1/(n:E1\n:E1)                      ::  [table(E1)].
table     :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [table(E1)].

telescope :=  n:E1                                  ::  [telescope(E1)].
telescope :=  n:E1/(n:E1\n:E1)                      ::  [telescope(E1)].
telescope :=  n:E1/(n:E1\n:E1)/(n:E1\n:E1)          ::  [telescope(E1)].


% Adjectives

blue      :=  n:E1/n:E1                             ::  [blue(E1)].
green     :=  n:E1/n:E1                             ::  [green(E1)].
red       :=  n:E1/n:E1                             ::  [red(E1)].


% Relative pronouns

who       :=  (n:E1\n:E1)/(s:E2\np:E1)              ::  [].
who       :=  (n:E1\n:E1)/(s:E2/np:E1)              ::  [].

which     :=  (n:E1\n:E1)/(s:E2\np:E1)              ::  [].
which     :=  (n:E1\n:E1)/(s:E2/np:E1)              ::  [].

that      :=  (n:E1\n:E1)/(s:E2\np:E1)              ::  [].
that      :=  (n:E1\n:E1)/(s:E2/np:E1)              ::  [].
```

```
% Prepositions

in      :=  (n:E1\n:E1)/np:E2                       ::  [in(E1,E2)].
on      :=  (n:E1\n:E1)/np:E2                       ::  [on(E1,E2)].
by      :=  (n:E1\n:E1)/np:E2                       ::  [by(E1,E2)].
with    :=  (n:E1\n:E1)/np:E2                       ::  [with(E1,E2)].

in      :=  ((s:E1\np:E2)\(s:E1\np:E2))/np:E3       ::  [in(E1,E3)].
on      :=  ((s:E1\np:E2)\(s:E1\np:E2))/np:E3       ::  [on(E1,E3)].
by      :=  ((s:E1\np:E2)\(s:E1\np:E2))/np:E3       ::  [by(E1,E3)].
with    :=  ((s:E1\np:E2)\(s:E1\np:E2))/np:E3       ::  [with(E1,E3)].


% Verbs

visits  :=  (s:E1\np:E2)/np:E3                      ::  [visit(E1,E2,E3)].
saw     :=  (s:E1\np:E2)/np:E3                      ::  [see(E1,E2,E3)].

runs    :=  s:E1\np:E2                              ::  [run(E1,E2)].
```

```
%%      A.3.  Context A:  Blocks and boxes


# box(box1).
# box(box2).
# box(box3).

# table(table1).
# table(table2).

# block(block1).
# block(block2).

# rabbit(rabbit1).
# rabbit(rabbit2).
# rabbit(rabbit3).

# hat(hat1).
# hat(hat2).

# green(box3).
# green(table2).
# green(block2).

# on(box1,table1).
# on(box3,table2).

# in(block1,box1).
# in(block2,box2).
# in(rabbit2,hat1).
# in(rabbit3,box1).


%  setupDomain
%  keeps track of the currently available contextual entities.  A
%  contextual entity is assumed to be any argument of a predicate
%  in the context.

setup:- retract(initialDomain(_)), fail.
setup:- findall(E, ( # P, P =.. [F|As], member(E,As) ), EntitiesList),
        listtoset(EntitiesList,Entities),
        assert(initialDomain(Entities)).

:- setup.
```

```
%%      A.3.  Context B:  People and places


# man(man1).
# man(man2).
# man(man3).
# man(man4).
# man(man5).

# woman(woman1).
# woman(woman2).

# boy(boy1).
# boy(boy2).

# girl(girl1).

# telescope(telescope1).
# telescope(telescope2).

# town(town1).
# town(town2).
# town(town3).
# town(town4).
# town(town5).

# village(village1).

# river(river1).
# river(river1).

# lake(lake1).

# visit(i1,man1,town1).
# visit(i2,man1,town2).
# visit(i3,man1,town3).
# visit(i4,man2,village1).
# visit(i5,man4,town3).
# visit(i6,man5,town4).
# visit(i7,woman1,town5).

# see(i10,woman1,boy1).
# see(i11,woman2,boy2).
# see(i12,girl1,woman1).
# see(i13,woman1,girl1).
# see(i14,woman2,girl1).

# run(i20,man3).
# run(i21,woman1).

# by(town1,river1).
# by(town2,river1).
```

```
# by(town3,lake1).
# by(town5,river2).
# by(village1,river2).

# with(boy1,telescope1).
# with(i14,telescope2).



%  setupDomain
%  keeps track of the currently available contextual entities.  A
%  contextual entity is assumed to be any argument of a predicate
%  in the context.

setup:- retract(initialDomain(_)), fail.
setup:- findall(E, ( # P, P =.. [F|As], member(E,As) ), EntitiesList),
        listtoset(EntitiesList,Entities),
        assert(initialDomain(Entities)).

:- setup.
```

```
%%      A.4.  Parser

%%      An implementation of the breadth-first chart parsing algorithm
%%      described in section 6.4 with an interface to the on-line
%%      processes of semantic evaluation as introduced in sections 5.2
%%      and 5.3.


%  The chart is represented as a collection of edges, each edge
%  consisting of a left vertex, a right vertex and a label.  An
%  edge is therefore represented as a three-part structure
%       edge(Lv,Rv,Label)
%  where the vertices Lv and Rv are represented as integers, and
%  Label has internal structure defined below.  Vertices and
%  labels are accessed by the following three operations.

leftVertex(V,edge(V,_,_)).
rightVertex(V,edge(_,V,_)).
label(L,edge(_,_,L)).


%  The label of an edge is represented as a four-slot structure
%       label(C,U,T,N)
%  where C is a syntactic/semantic category, U is a list of closure
%  constraints, T is a list of network constraints, and N is the
%  network itself, made up of arcs and value domains.  These
%  respective label attributes are accessed by the following four
%  operations.

cat(C,label(C,_,_,_)).
u_trans(U,label(_,U,_,_)).
n_trans(T,label(_,_,T,_)).
network(N,label(_,_,_,N)).


%  createEdge(Edge,Lv,Rv,Label)
%  creates a chart edge template Edge with left vertex Lv,
%  right vertex Rv, and Label.

createEdge(Edge,Lv,Rv,Label):-
        label(Label,Edge),
        leftVertex(Lv,Edge),
        rightVertex(Rv,Edge).


%  createLabel(L,C,U,T,N)
%  creates an edge label template L with category C, closure
%  constraints U, network constraints T and constraint network N.

createLabel(L,C,U,T,N):-
        cat(C,L),
        u_trans(U,L),
```

```
        n_trans(T,L),
        network(N,L).


%  Set the type of category we are looking for: an NP associated with
%  a semantic variable V.  Set the initial chart vertex to 0.

startSymbol(np:V).
startVertex(0).


%  successfulEdge(E,EndVertex)
%  defines the properties of a successful chart edge E.  E must be
%  labelled with the start symbol Cat, have a left vertex
%  StartVertex (the first vertex of the chart) and right vertex
%  EndVertex (the last vertex of the chart).

successfulEdge(E,EndVertex):-
        startSymbol(Cat),
        label(L,E),
        cat(Cat,L),
        startVertex(StartVertex),
        leftVertex(StartVertex,E),
        rightVertex(EndVertex,E).


%  parse
%  clears up any chart edges in the database from previous parsings,
%  reads the input string into a buffer, calls the main parsing
%  procedure and prints the results.

parse:- clearchart,
        startVertex(StartVertex),
        readInput(String),
        parse(StartVertex,String,Parses),
        writeFinal(Parses).


%  parse(EndVertex,String,Parses)
%  iterates through the input buffer from left to right, incrementing
%  the vertex counter and calling read/3 as each word is read.  When
%  the buffer is empty, the variable Parses is instantiated to the set
%  of successful chart edges in the database.

parse(EndVertex,[],Parses):-
        successfulEdge(E,EndVertex),
        findall(E,E,Parses).
parse(EndVertex,[Word|String],Parses):-
        NewEndVertex is EndVertex + 1,
        writeReading(Word,EndVertex,NewEndVertex),
        read(Word,EndVertex,NewEndVertex),
        parse(NewEndVertex,String,Parses).
```

```
%  read(Word,LeftVertex,RightVertex)
%  is the first of two core parsing procedures, the second of which is
%  add/1.  It finds a lexical entry for Word, splits the lexical
%  constraints into closure constraints UTrans and network constraints
%  NTrans.  It determines the constraint Network pertaining to these
%  NTrans, and then creates and instantiates edge and label structures
%  appropriately.  The edge is then incorporated into the chart using
%  add/1, but since add/1 always fails, so will the first clause of
%  read/3.  Hence the second clause, which will be found only once
%  all lexical entries for Word have been discovered.

read(Word,Lv,Rv):-
        Word := Cat::Constraints,
        split(Constraints,UTrans,NTrans),
        createNetwork(NTrans,Network),
        createLabel(Label,Cat,UTrans,NTrans,Network),
        createEdge(E,Lv,Rv,Label),
        add(E).
read(_,_,_).


%  add(E2)
%  asserts the edge E2 to the database and then looks for any existing
%  edges E1 to the immediate left of E2.  For each such E1, an attempt
%  is made to syntactically rewrite the categories of these edges, C1
%  and C2, to a reduced category C0.  If this is possible, the closure
%  constraints (U1 and U2) and network constraints (T1 and T2) of the
%  two constituents are combined, and an attempt is made to merge the
%  corresponding constraint networks N1 and N2.  If the Type of
%  combinatory rule used was function application, closure constraints
%  are evaluated and the new closure list U0 set to [].  If all this
%  succeeds, an edge structure is created with the appropriate
%  slot-fillers and, if a similar edge is not already on the chart,
%  add/1 is called recursively with the new addition.
%  Because add/1 moves leftwards in the chart, and because the chart
%  starts at a fixed point, add/1 ultimately always fails.  This ensures
%  that all relevant left adjacent edges E1 will be found for a given
%  E2, and that all syntactic rules will be discovered for a given C1
%  and C2.

add(E2):-
        assertz(E2),
        writeNewEdge(E2),
        !,
        leftAdjacent(E1,E2),
        label(L1,E1),
        label(L2,E2),
        cat(C1,L1),
        cat(C2,L2),
        u_trans(U1,L1),
```

```
        u_trans(U2,L2),
        n_trans(T1,L1),
        n_trans(T2,L2),
        network(N1,L1),
        network(N2,L2),
        synRule(Type,C1,C2,C0),
        semRule(Type,U1::T1,U2::T2,U::T0),
        mergeNetworks(N1,N2,N0),
        (Type == a -> (enforceClosures(U,N0), U0 = []); U0 = U),
        createLabel(L0,C0,U0,T0,N0),
        leftVertex(Lv,E1),
        rightVertex(Rv,E2),
        createEdge(E0,Lv,Rv,L0),
        \+ redundant(E0),
        add(E0).


%  leftAdjacent(E,A)
%  defines what it means for an edge E to be left-adjacent to an edge A:
%  if the right vertex of E equals the left vertex of A, and if E exists
%  in the chart.

leftAdjacent(E,A):-
        leftVertex(Lv,A),
        rightVertex(Lv,E),
        E.              % E is in the chart if it's in the database


%  redundant(E)
%  defines what it means for an edge structure E to be redundant.
%  E is redundant if some existing edge Q spans same vertices and
%  their categories, closure constraints and network constraints are
%  functionally equivalent.  (It is unnecessary to also check the state
%  of evaluation in the constraint networks since all functionally equivalent
%  constraint problems will evaluate similarly.)

redundant(E):-
        createEdge(E,Lv,Rv,Le),         % use createEdge backwards for
        createEdge(Q,Lv,Rv,Lq),         % quick access of E's attributes
        Q,
        cat(Ce,Le),
        cat(Cq,Lq),
        u_trans(Ue,Le),
        u_trans(Uq,Lq),
        n_trans(Te,Le),
        n_trans(Tq,Lq),
        equivalent([Ce,Ue,Te],[Cq,Uq,Tq]).


%  equivalent(A,B)
%  is true when A and B can be unified, and, moreover, when they can still
%  be unified after each distinct variable in A and B has been bound to
```

```
%  a unique ground term.  In terms of equality, then, equivalent/2 is
%  stronger than term unification ("=") but weaker than identity ("==").

equivalent(A,B):-
        \+ \+ (A = B),
        \+ \+ (numbervars(A,0,N),
               numbervars(B,0,N),
               A = B).


%  split(C,U,T)
%  takes a lexical constraint list C and divides it into closure constraints
%  U and ordinary network constraints T, by virtue of the "*" prefix which
%  distinguishes any closure constraints in the lexical constraint list.

split([],[],[]).
split([*U|Sems],[U|Us],Ts):-
        split(Sems,Us,Ts).
split([T|Sems],Us,[T|Ts]):-
        T \= *U,
        split(Sems,Us,Ts).
```

```
%%      A.5.  Evaluation Procedures

%%      Includes the adaptation of Mackworth's (1977b) network
%%      consistency algorithm NC, as described in sections 4.4
%%      and 5.2, and the mechanism for evaluating closure
%%      constraints, as described in section 5.3.3.


%%      Networks and network objects

%  A constraint network consists of two sets: a set of arcs (derived
%  from the input constraints) and a set of domains (derived from the
%  context via consistency operations).  We therefore represent a
%  constraint network as a structure with two slots:
%       net(TSet,DSet)
%  where TSet is the set of arcs and DSet is the set of value domains.
%
%  The following two operations can be used to access these parts of
%  a network, or create a new network template.

tSet(TSet,net(TSet,_)).
dSet(DSet,net(_,DSet)).


%  For convenience, each arc in TSet is represented as a triple
%       t(X,R,M)
%  where X is the variable at the initial node of the arc, R is a
%  constraint, and M is the set of variables other than X which are
%  directly constrained by R.  Given this implementation, we will refer
%  to arcs as "triples".
%
%  The following three operations access these parts of a triple, or
%  build a triple template.

tvar(X,t(X,_,_)).
rel(R,t(_,R,_)).
other(M,t(_,_,M)).


%  Each domain in DSet is represented as a structure with two slots
%       d(X,D)
%  where X is a variable and D is the domain of X.
%
%  The following two operations access X and D in a domain structure
%  or build a new template.

dvar(X,d(X,_)).
dom(D,d(_,D)).


%%      Creating, accessing and changing parts of the network
```

```
%  initialNet(Net)
%  creates a new network template Net with empty TSet and DSet.

initialNet(Net):- tSet([],Net), dSet([],Net).


%  createDomain(X,DSet,NewDSet)
%  creates a template domain structure with variable X and a sets
%  its domain to the set of all contextual entities.

createDomain(X,DSet,[DStruct|DSet]):-
        dvar(X,DStruct),
        dom(Dx,DStruct),
        initialDomain(Dx).


%  fetchDomain(Dx,X,DSet)
%  returns the domain Dx for the variable X in DSet.

fetchDomain(Dx,X1,[DStruct|DSet]):-
        dvar(X2,DStruct),
        X1 == X2,
        !,
        dom(Dx,DStruct).
fetchDomain(Dx,X,[_|DSet]):- fetchDomain(Dx,X,DSet).


%  changeDomain(X,OldDx,OldDSet,NewDx,NewDSet)
%  replaces the old domain of X, OldDx, by NewDx, so turning OldDSet
%  into NewDSet.

changeDomain(X1,OldDx,[OldDStruct|DSet],NewDx,[NewDStruct|DSet]):-
        dvar(X2,OldDStruct),
        X1 == X2,
        !,
        dom(OldDx,OldDStruct),
        dvar(X2,NewDStruct),
        dom(NewDx,NewDStruct).
changeDomain(X,OldDx,[Dy|DSet],NewDx,[Dy|NewDSet]):-
        changeDomain(X,OldDx,DSet,NewDx,NewDSet).


%%      Evaluation routines

%  enforceClosures(ClosureList,Network)
%  enforces each closure constraint in ClosureList with respect to the
%  domains DSet in Network.  It can evaluate closure constraints of any
%  arity M, although the only closure constraints implemented to date
%  are constraints of uniqueness with arity 1.

enforceClosures(ClosureList,Network):-
        dSet(DSet,Network),
```

```
        forall(member(U,ClosureList), (functor(U,_,M), satisfyU(U,DSet,M))).


%  satisfyU(U,DSet,M)
%  instantiates each of the M semantic variables in U to the
%  appropriate domain in DSet and, when U is fully instantiated,
%  tries to prove it.

satisfyU(U,_,0):- call(U).
satisfyU(U,DSet,M):-
        M > 0,
        arg(M,U,X),
        fetchDomain(X,X,DSet),
        K is M-1,
        satisfyU(U,DSet,K).


%  An extension is unique if its domain, represented as a list,
%  contains one item exactly.

unique([_]).


%  createNetwork(Constraints,Network)
%  creates a new network structure Net and then incorporates each
%  constraint in the list Constraints into Net, to yield Network.

createNetwork(Constraints,Network):-
        initialNet(Net),
        constraint(Constraints,Net,Network).


%  constraint(Constraints,Net,NewNet)
%  adds each of the Constraints to Net to produce NewNet.
%  Consistency is maintained after each addition to the network.

constraint([],Net,Net):- !.
constraint([H|T],Net,NewNet):-
        !,
        constraint(H,Net,Net1),
        constraint(T,Net1,NewNet).
constraint(R,Net,NewNet):-
        makeTriples(R,Q,Net,Net1),
        propagate(Q,Net1,NewNet).


%  makeTriples(R,NewTriples,Net,NewNet)
%  updates the network Net with the NewTriples derived from the
%  constraint R to produce NewNet.  It also
%  creates an initial value domain for any variables occuring in R
%  which are not already in Net, and updates the DSet appropriately.
%  If R has arity 0, it is evaluated here.
```

```
makeTriples(R,NewTriples,Net,NewNet):-
        tSet(TSet,Net),
        tSet(NewTSet,NewNet),
        dSet(DSet,Net),
        dSet(NewDSet,NewNet),
        R =.. [P|As],
        makeTriples1(R,As,[],NewTriples,DSet,NewDSet),
        (  As = [], #R, NewTSet = TSet
         ; As = [_], NewTSet = TSet
         ; As = [_|_], append(TSet,NewTriples,NewTSet) ),
        !.


%  makeTriples1(R,M1,M2,Ts,DSet,NewDSet)
%  derives a set of triples Ts from the constraint R, and creates an
%  initial value domain for any variables in R which are not already
%  associated with a domain in DSet, thus producing a new set of
%  domains DSet.  M1 and M2 are complementary lists of variables in R,
%  used to produce each triple.

makeTriples1(_,[],_,[],DSet,DSet).
makeTriples1(R,[X|M1],M2,[T|Ts],DSet,NewDSet):-
        tvar(X,T),
        rel(R,T),
        other(M,T),
        append(M1,M2,M),
        (fetchDomain(_,X,DSet) -> NewDSet1 = DSet
         ; createDomain(X,DSet,NewDSet1)),
        makeTriples1(R,M1,[X|M2],Ts,NewDSet1,NewDSet).


%  mergeNetworks(N1,N2,N0)
%  merges two networks, N1 and N2, to form a new network N0.  The set
%  of triples of N1, TSet1, and those of N2, TSet2, are appended to
%  form a composite triple set, TSet.  The component sets of domains,
%  DSet1 and DSet2, are merged into a new set of domains, DSet.
%  The set of triples Triples in TSet which are directly affected by
%  this merger then form the initial queue for propagation.
%  If successful, the propagation phase returns a consistent network N0.

mergeNetworks(N1,N2,N0):-
        tSet(TSet1,N1),
        tSet(TSet2,N2),
        tSet(TSet,N),
        append(TSet1,TSet2,TSet),
        dSet(DSet1,N1),
        dSet(DSet2,N2),
        dSet(DSet,N),
        merge(DSet1,DSet2,DSet,TSet,Triples),
        propagate(Triples,N,N0).
```

```
%  merge(DSetA,DSetB,DSet,TSet,Triples)
%  merges two sets of domains, DSetA and DSetB, into DSet.
%  For each variable-domain pair <X,Da> in DSetA, if the same
%  variable X is also linked to a domain Db in DSetB, then Da and
%  Db are intersected to form Dx.  If Dx == [], the procedure fails.
%  The set of triples in TSet which are directly affected by any such
%  intersections are returned in the set Triples.

merge([],DSetB,DSetB,_,[]).
merge([DStructA|DSetA],DSetB,DSet,TSet,Triples):-
        dvar(X,DStructA),
        dom(Da,DStructA),
        changeDomain(X,Db,DSetB,Dx,DSetB1),
        !,
        intersect(Da,Db,Dx),
        Dx \== [],
        tvar(X,T),                      % because R is unbound in T, all
        connected(Triples1,T,TSet),     % relevant Triples1 are found
        merge(DSetA,DSetB1,DSet,TSet,Triples2),
        union(Triples1,Triples2,Triples).
merge([DStructA|DSetA],DSetB,[DStructA|DSet],TSet,Triples):-
        merge(DSetA,DSetB,DSet,TSet,Triples).


%  propagate(Queue,Net,FinalNet)
%  takes the first triple T from Queue and revises the current domain
%  of its initial node to produce a new domain NewD.  If NewD == [], the
%  procedure fails.  If NewD is otherwise different to the original
%  (iff revise/4 holds), then any triples Ts which might be affected
%  by this change are added to the queue for the next phase of
%  propagation.

propagate([],Net,Net).
propagate([T|Q],Net,FinalNet):-
        tSet(TSet,Net),
        dSet(DSet,Net),
        tSet(TSet,NewNet),
        dSet(NewDSet,NewNet),
        revise(T,NewD,DSet,NewDSet),
        !,
        NewD \== [],
        connected(Ts,T,TSet),
        union(Q,Ts,NewQ),
        propagate(NewQ,NewNet,FinalNet).
propagate([_|Q],Net,FinalNet):-
        propagate(Q,Net,FinalNet).


%  revise(T,NewDx,DSet,NewDSet)
%  fetches from DSet the current domain Dx of the initial variable X
%  of triple T and filters Dx with respect to the constraint R in T.
```

```
%  If the filtered domain NewDx is the same as the original Dx, the
%  procedure fails.

revise(T,NewDx,DSet,NewDSet):-
        tvar(X,T),
        rel(R,T),
        other(M,T),
        fetchDomain(Dx,X,DSet),
        filter(X,R,M,Dx,NewDx,DSet),
        Dx \== NewDx,
        changeDomain(X,Dx,DSet,NewDx,NewDSet).


%  filter(X,R,M,Dx,NewDx,DSet)
%  instantiates the variable X to each of the entities in its domain Dx,
%  and for each instantiation tries to satisfy the constraint R with
%  respect to some instantiation of the remaining variables M in R
%  from their respective domains in DSet.  NewDx consists of those
%  entities in Dx for which R can be satisfied in this way.

filter(_,_,_,[],[],DSet).
filter(X,R,M,[E|Dx],[E|NewDx],DSet):-
        \+ \+ (X = E,
               satisfyP(R,M,DSet)),
        !,
        filter(X,R,M,Dx,NewDx,DSet).
filter(X,R,M,[_|Dx],NewDx,DSet):-
        filter(X,R,M,Dx,NewDx,DSet).


%  satisfyP(R,M,DSet)
%  attempts to find some instantiation of the remaining variables M in
%  the constraint R from their respective value domains in DSet such
%  that R occurs in the context.

satisfyP(R,[],DSet):- #R.
satisfyP(R,[Y|M],DSet):-
        \+ \+ (#R),             % prove the (unground) relation at every
        !,                      % stage of instantiation, for efficiency
        fetchDomain(Dy,Y,DSet),
        member(Y,Dy),
        satisfyP(R,M,DSet).


%  connected(Ts,T,TSet)
%  computes the set of triples Ts from the entire triple set TSet which
%  need to be reconsidered as a result of refining triple T.  See section
%  4.4 for a fuller explanation.

connected([],_,[]).
connected([Tk|Triples],T,[Tk|TSet]):-
        tvar(X,T),
```

```
        rel(R,T),
        rel(Rk,Tk),
        other(Mk,Tk),
        Rk \== R,
        membereq(X,Mk),          % membereq/2 is just like member/2 except
        !,                       % that it tests for membership with "=="
        connected(Triples,T,TSet).
connected(Triples,T,[_|TSet]):-
        connected(Triples,T,TSet).
```

%% A.6. Test Runs

```
/*****************************************************************************

Below are eight example traces of the program: the first three traces
use Context A in A.3, while the remaining five use Context B.
Context switching is done explicitly using the procedure "loadContext".

The first two examples are traced in detail, by setting the flag "loud".
This prints out the features of each chart edge as it is added to the
database.  Edges are printed in the following format:

        LeftVertex --> RightVertex
           Category
           ClosureConstraints
           NetworkConstraints
           Domains

Because a large number of chart edges get added during processing, the
last six examples operate in "quiet" mode; this does not print chart edges,
instead just reporting any successful parses at the end of processing.

*****************************************************************************/


?- [pload].
ops reconsulted
parser reconsulted
grammar reconsulted
evaluation reconsulted
util reconsulted
input reconsulted
output reconsulted
yes

pload consulted
yes

?- loadContext.
|: contextA.
yes

?- loud.
yes

?- parse.
|: the green table

Reading "the" from 0 to 1

0 --> 1
```

```
   np : E1 / n : E1
   [unique(E1)]
   []
   []
0 --> 1
   s : E1 / (s : E1 \ np : E2) / n : E2
   [unique(E2)]
   []
   []

Reading "green" from 1 to 2

1 --> 2
   n : E1 / n : E1
   []
   [green(E1)]
   [Dom(E1) = {box3,table2,block2}]
0 --> 2
   np : E1 / n : E1
   [unique(E1)]
   [green(E1)]
   [Dom(E1) = {box3,table2,block2}]
0 --> 2
   s : E1 / (s : E1 \ np : E2) / n : E2
   [unique(E2)]
   [green(E2)]
   [Dom(E2) = {box3,table2,block2}]

Reading "table" from 2 to 3

2 --> 3
   n : E1
   []
   [table(E1)]
   [Dom(E1) = {table1,table2}]
1 --> 3
   n : E1
   []
   [green(E1), table(E1)]
   [Dom(E1) = {table2}]
0 --> 3
   np : E1
   []
   [green(E1), table(E1)]
   [Dom(E1) = {table2}]
0 --> 3
   s : E1 / (s : E1 \ np : E2)
   []
   [green(E2), table(E2)]
   [Dom(E2) = {table2}]
2 --> 3
   n : E1 / (n : E1 \ n : E1)
```

```
   []
   [table(E1)]
   [Dom(E1) = {table1,table2}]
1 --> 3
   n : E1 / (n : E1 \ n : E1)
   []
   [green(E1), table(E1)]
   [Dom(E1) = {table2}]
0 --> 3
   np : E1 / (n : E1 \ n : E1)
   [unique(E1)]
   [green(E1), table(E1)]
   [Dom(E1) = {table2}]
0 --> 3
   s : E1 / (s : E1 \ np : E2) / (n : E2 \ n : E2)
   [unique(E2)]
   [green(E2), table(E2)]
   [Dom(E2) = {table2}]
2 --> 3
   n : E1 / (n : E1 \ n : E1) / (n : E1 \ n : E1)
   []
   [table(E1)]
   [Dom(E1) = {table1,table2}]


PARSES ARE:

Parse 1:
0 --> 3
   np : E1
   []
   [green(E1), table(E1)]
   [Dom(E1) = {table2}]


Total number of parses: 1
yes

?- parse.
|: the rabbit in the hat

Reading "the" from 0 to 1

0 --> 1
   np : E1 / n : E1
   [unique(E1)]
   []
   []
0 --> 1
   s : E1 / (s : E1 \ np : E2) / n : E2
   [unique(E2)]
   []
```

```
   []

Reading "rabbit" from 1 to 2

1 --> 2
   n : E1
   []
   [rabbit(E1)]
   [Dom(E1) = {rabbit1,rabbit2,rabbit3}]
1 --> 2
   n : E1 / (n : E1 \ n : E1)
   []
   [rabbit(E1)]
   [Dom(E1) = {rabbit1,rabbit2,rabbit3}]
0 --> 2
   np : E1 / (n : E1 \ n : E1)
   [unique(E1)]
   [rabbit(E1)]
   [Dom(E1) = {rabbit1,rabbit2,rabbit3}]
0 --> 2
   s : E1 / (s : E1 \ np : E2) / (n : E2 \ n : E2)
   [unique(E2)]
   [rabbit(E2)]
   [Dom(E2) = {rabbit1,rabbit2,rabbit3}]
1 --> 2
   n : E1 / (n : E1 \ n : E1) / (n : E1 \ n : E1)
   []
   [rabbit(E1)]
   [Dom(E1) = {rabbit1,rabbit2,rabbit3}]

Reading "in" from 2 to 3

2 --> 3
   n : E1 \ n : E1 / np : E2
   []
   [in(E1, E2)]
   [Dom(E2) = {box2,hat1,box1}, Dom(E1) = {block1,block2,rabbit2,rabbit3}]
1 --> 3
   n : E1 / np : E2
   []
   [rabbit(E1), in(E1, E2)]
   [Dom(E2) = {hat1,box1}, Dom(E1) = {rabbit2,rabbit3}]
0 --> 3
   np : E1 / np : E2
   [unique(E1)]
   [rabbit(E1), in(E1, E2)]
   [Dom(E2) = {hat1,box1}, Dom(E1) = {rabbit2,rabbit3}]
0 --> 3
   s : E1 / (s : E1 \ np : E2) / np : E3
   [unique(E2)]
   [rabbit(E2), in(E2, E3)]
   [Dom(E3) = {hat1,box1}, Dom(E2) = {rabbit2,rabbit3}]
```

```
1 --> 3
   n : E1 / (n : E1 \ n : E1) / np : E2
   []
   [rabbit(E1), in(E1, E2)]
   [Dom(E2) = {hat1,box1}, Dom(E1) = {rabbit2,rabbit3}]
2 --> 3
   s : E1 \ np : E2 \ (s : E1 \ np : E2) / np : E3
   []
   [in(E1, E3)]
   [Dom(E3) = {box2,hat1,box1}, Dom(E1) = {block1,block2,rabbit2,rabbit3}]

Reading "the" from 3 to 4

3 --> 4
   np : E1 / n : E1
   [unique(E1)]
   []
   []
2 --> 4
   n : E1 \ n : E1 / n : E2
   [unique(E2)]
   [in(E1, E2)]
   [Dom(E2) = {box2,hat1,box1}, Dom(E1) = {block1,block2,rabbit2,rabbit3}]
1 --> 4
   n : E1 / n : E2
   [unique(E2)]
   [rabbit(E1), in(E1, E2)]
   [Dom(E2) = {hat1,box1}, Dom(E1) = {rabbit2,rabbit3}]
0 --> 4
   np : E1 / n : E2
   [unique(E1), unique(E2)]
   [rabbit(E1), in(E1, E2)]
   [Dom(E2) = {hat1,box1}, Dom(E1) = {rabbit2,rabbit3}]
0 --> 4
   s : E1 / (s : E1 \ np : E2) / n : E3
   [unique(E2), unique(E3)]
   [rabbit(E2), in(E2, E3)]
   [Dom(E3) = {hat1,box1}, Dom(E2) = {rabbit2,rabbit3}]
1 --> 4
   n : E1 / (n : E1 \ n : E1) / n : E2
   [unique(E2)]
   [rabbit(E1), in(E1, E2)]
   [Dom(E2) = {hat1,box1}, Dom(E1) = {rabbit2,rabbit3}]
2 --> 4
   s : E1 \ np : E2 \ (s : E1 \ np : E2) / n : E3
   [unique(E3)]
   [in(E1, E3)]
   [Dom(E3) = {box2,hat1,box1}, Dom(E1) = {block1,block2,rabbit2,rabbit3}]
3 --> 4
   s : E1 / (s : E1 \ np : E2) / n : E2
   [unique(E2)]
   []
```

```
   []

Reading "hat" from 4 to 5

4 --> 5
   n : E1
   []
   [hat(E1)]
   [Dom(E1) = {hat2,hat1}]
2 --> 5
   n : E1 \ n : E1
   []
   [in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
1 --> 5
   n : E1
   []
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
0 --> 5
   np : E1
   []
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
0 --> 5
   s : E1 / (s : E1 \ np : E2)
   []
   [rabbit(E2), in(E2, E3), hat(E3)]
   [Dom(E2) = {rabbit2}, Dom(E3) = {hat1}]
1 --> 5
   n : E1 / (n : E1 \ n : E1)
   []
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
0 --> 5
   np : E1 / (n : E1 \ n : E1)
   [unique(E1)]
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
0 --> 5
   s : E1 / (s : E1 \ np : E2) / (n : E2 \ n : E2)
   [unique(E2)]
   [rabbit(E2), in(E2, E3), hat(E3)]
   [Dom(E2) = {rabbit2}, Dom(E3) = {hat1}]
2 --> 5
   s : E1 \ np : E2 \ (s : E1 \ np : E2)
   []
   [in(E1, E3), hat(E3)]
   [Dom(E1) = {rabbit2}, Dom(E3) = {hat1}]
4 --> 5
   n : E1 / (n : E1 \ n : E1)
   []
```

```
   [hat(E1)]
   [Dom(E1) = {hat2,hat1}]
3 --> 5
   np : E1 / (n : E1 \ n : E1)
   [unique(E1)]
   [hat(E1)]
   [Dom(E1) = {hat2,hat1}]
2 --> 5
   n : E1 \ n : E1 / (n : E2 \ n : E2)
   [unique(E2)]
   [in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
1 --> 5
   n : E1 / (n : E2 \ n : E2)
   [unique(E2)]
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
0 --> 5
   np : E1 / (n : E2 \ n : E2)
   [unique(E1), unique(E2)]
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
0 --> 5
   s : E1 / (s : E1 \ np : E2) / (n : E3 \ n : E3)
   [unique(E2), unique(E3)]
   [rabbit(E2), in(E2, E3), hat(E3)]
   [Dom(E2) = {rabbit2}, Dom(E3) = {hat1}]
1 --> 5
   n : E1 / (n : E1 \ n : E1) / (n : E2 \ n : E2)
   [unique(E2)]
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]
2 --> 5
   s : E1 \ np : E2 \ (s : E1 \ np : E2) / (n : E3 \ n : E3)
   [unique(E3)]
   [in(E1, E3), hat(E3)]
   [Dom(E1) = {rabbit2}, Dom(E3) = {hat1}]
3 --> 5
   s : E1 / (s : E1 \ np : E2) / (n : E2 \ n : E2)
   [unique(E2)]
   [hat(E2)]
   [Dom(E2) = {hat2,hat1}]
4 --> 5
   n : E1 / (n : E1 \ n : E1) / (n : E1 \ n : E1)
   []
   [hat(E1)]
   [Dom(E1) = {hat2,hat1}]


PARSES ARE:

Parse 1:
```

```
0 --> 5
   np : E1
   []
   [rabbit(E1), in(E1, E2), hat(E2)]
   [Dom(E1) = {rabbit2}, Dom(E2) = {hat1}]


Total number of parses: 1
yes

?- quiet.
yes

?- parse.
|: the block in the box on the table

Reading "the" from 0 to 1


Reading "block" from 1 to 2


Reading "in" from 2 to 3


Reading "the" from 3 to 4


Reading "box" from 4 to 5


Reading "on" from 5 to 6


Reading "the" from 6 to 7


Reading "table" from 7 to 8



PARSES ARE:

Parse 1:
0 --> 8
   np : E1
   []
   [block(E1), in(E1, E2), box(E2), on(E2, E3), table(E3)]
   [Dom(E1) = {block1}, Dom(E2) = {box1}, Dom(E3) = {table1}]


Total number of parses: 1
```

```
yes

?- loadContext.
|: contextB.
yes

?- parse.
|: the man who runs

Reading "the" from 0 to 1


Reading "man" from 1 to 2


Reading "who" from 2 to 3


Reading "runs" from 3 to 4



PARSES ARE:

Parse 1:
0 --> 4
   np : E1
   []
   [man(E1), run(E2, E1)]
   [Dom(E1) = {man3}, Dom(E2) = {i20}]


Total number of parses: 1
yes

?- parse.
|: a woman that the girl saw

Reading "a" from 0 to 1


Reading "woman" from 1 to 2


Reading "that" from 2 to 3


Reading "the" from 3 to 4


Reading "girl" from 4 to 5
```

```
Reading "saw" from 5 to 6



PARSES ARE:

Parse 1:
0 --> 6
   np : E1
   []
   [woman(E1), girl(E2), see(E3, E2, E1)]
   [Dom(E1) = {woman1}, Dom(E2) = {girl1}, Dom(E3) = {i12}]


Total number of parses: 1
yes

?- parse.
|: the man who visits a town by the river

Reading "the" from 0 to 1


Reading "man" from 1 to 2


Reading "who" from 2 to 3


Reading "visits" from 3 to 4


Reading "a" from 4 to 5


Reading "town" from 5 to 6


Reading "by" from 6 to 7


Reading "the" from 7 to 8


Reading "river" from 8 to 9



PARSES ARE:

Parse 1:
```

```
0 --> 9
   np : E1
   []
   [man(E1), visit(E2, E1, E3), town(E3), by(E3, E4), river(E4)]
   [Dom(E1) = {man1}, Dom(E2) = {i1,i2}, Dom(E3) = {town1,town2}, Dom(E4)
        = {river1}]


Total number of parses: 1
yes

?- parse.
|: the woman who saw the boy with the telescope

Reading "the" from 0 to 1


Reading "woman" from 1 to 2


Reading "who" from 2 to 3


Reading "saw" from 3 to 4


Reading "the" from 4 to 5


Reading "boy" from 5 to 6


Reading "with" from 6 to 7


Reading "the" from 7 to 8


Reading "telescope" from 8 to 9



PARSES ARE:

Parse 1:
0 --> 9
   np : E1
   []
   [woman(E1), see(E2, E1, E3), boy(E3), with(E3, E4), telescope(E4)]
   [Dom(E1) = {woman1}, Dom(E2) = {i10}, Dom(E3) = {boy1}, Dom(E4) = {telescop
        e1}]
```

```
Total number of parses: 1
yes

?- parse.
|: the woman who saw the girl with a telescope

Reading "the" from 0 to 1


Reading "woman" from 1 to 2


Reading "who" from 2 to 3


Reading "saw" from 3 to 4


Reading "the" from 4 to 5


Reading "girl" from 5 to 6


Reading "with" from 6 to 7


Reading "a" from 7 to 8


Reading "telescope" from 8 to 9



PARSES ARE:

Parse 1:
0 --> 9
   np : E1
   []
   [woman(E1), see(E2, E1, E3), girl(E3), with(E2, E4), telescope(E4)]
   [Dom(E1) = {woman2}, Dom(E3) = {girl1}, Dom(E2) = {i14}, Dom(E4) = {telesco
        pe2}]


Total number of parses: 1
yes
```

Appendix B

Published Work

A copy of the following paper by the author is included.

Incremental Interpretation and Combinatory Categorial Grammar. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, August 1987, pp.661-663.

Incremental Interpretation and Combinatory Categorial Grammar

Nicholas J. Haddock
Department of Artificial Intelligence
and Centre for Cognitive Science,
University of Edinburgh.

April 1987

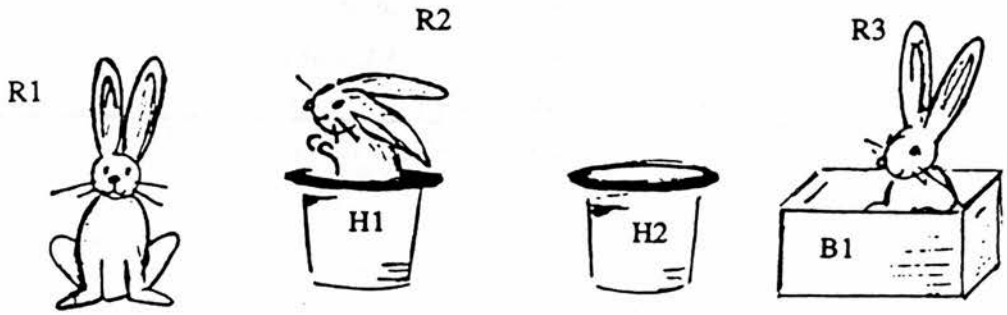
ABSTRACT

The paper is organised around a particular problem of anaphoric definite reference. A solution to this problem requires sentence processing which allows semantic representations to be evaluated *incrementally*, as a phrase is read from left to right. The paper shows how a proposal made by Mellish (1985), to regard incremental semantic evaluation as a constraint satisfaction task, can provide a natural solution to the problem of reference. The incorporation of Combinatory Categorial Grammar allows a straightforward relationship to be stated between the incrementally assembled syntactic and semantic representations.

This paper appears in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, 1987.

I INTRODUCTION

The concern of this paper is to show how a computational model of incremental interpretation suggests a solution to a particular problem of definite reference. Suppose we have a context in which there are three rabbits (R1, R2, and R3), two hats (H1 and H2) and one box (B1), where one rabbit is in a hat and one is in the box:



If both speaker and hearer are aware of this context, the second rabbit, R2, can be referred to by means of the following complex NP:

(1) the rabbit in the hat

This expression can be uttered without any previous talk of rabbits or hats. What is interesting about it is the use of the definite determiner in the inner, simple NP, *the hat*. As a whole, the expression in (1) sounds perfectly natural, and so suggests that a unique hat is identifiable to the hearer. But given that there are two hats, and not one, in the scene, why is the definite permissible? Any compositional accounts of NP semantics, including those of Winograd (1972) and Hirst (1983), would judge (1) to be infelicitous. Such theories would require the somewhat convoluted version in (2), if a definite is to be used at all to specify the hat.

(2) the rabbit in the hat with a rabbit in it

A semantics organised around the criterion of incremental interpretation should have less trouble with (1). If we assume that a hearer *incrementally* evaluates a semantic representation — after each word, say — the empty hat in the scene will never really be considered a viable candidate for the inner NP. When the word *rabbit* is reached, a hearer can collect together in his mind the set of rabbits in the context. After the preposition, this set can be refined to contain only rabbits which are *in* something and, most importantly, the hearer

can start thinking about another set of objects, those which have rabbits in them. There is only one hat in this new set and so by the time the inner NP is processed a definite determiner sounds natural. How can we pin this idea down in more explicit, computational terms?

II REFERENCE AND CONSTRAINT SATISFACTION

Mellish (1985) has proposed that the general process of incremental reference evaluation, sketched above, can be directly expressed by a constraint satisfaction algorithm of the kind presented by Mackworth (1977). Consider the following predicates, which may arise from the fragment *the rabbit in* (ignoring the determiner):

(3) rabbit(e1)
in(e1,e2)

Constraint satisfaction involves determining, for each of the variables e1 and e2, the entities in the discourse which satisfy the predicates, and collecting these into candidate sets for each variable. (We shall refer to these variables as *extension variables*.) Given the context provided for (1) above, this operation will label e1 with the set {R2,R3} and e2 with {H1,B1}. The advantage of the scheme is at once clear: satisfying the constraints in (3) determines a candidate set for e2 as well as e1, and so the reference of any NP which follows the fragment is already partially-evaluated. This offers a natural solution to the problem of non-compositionality exhibited in (1); a solution which is difficult to obtain in compositional semantic systems.

The following treatment of our problematic phrase illustrates two significant improvements on Mellish's own account of anaphoric definite reference. First, the phrase is evaluated strictly word-by-word, in left-to-right order; there is linguistic, psychological and intuitive support for this general position (Steedman (1987)). In contrast, Mellish's system would evaluate the two nouns before considering the preposition which relates them. Second, the relationship between syntactic and semantic descriptions is made explicit. Mellish himself points out that his system "lacked precision" on this issue (Mellish 1985:114).

III THE MODEL

The model sees syntax as controlling the way in which semantic representations are combined. And the goal is to provide an incremental semantics, so that a phrase receives its interpretation in stages, after each word is read. If this process is to accord with the rules of syntax, then the grammar too must be incremental.

A framework which might be helpful in this respect is Combinatory Categorial Grammar (Ades and Steedman (1982), Steedman (1985)). Like all categorial grammars, Combinatory Grammar employs a rule of *function application* to combine functions and arguments:

(4) X/Y + Y → X

Steedman adds further combinatory rules, including *function composition:*

(5) X/Y + Y/Z → X/Z

As we shall see, function composition is especially relevant to the issue of incrementation because it allows "incomplete" constituents, such as *the rabbit in* or *you can believe that I like,* to be elegantly described.

How can a constraint-based semantics be tailored to a system of combinatory grammar? Suppose we associate a constraint with each nominal word and an extension variable with its category. The noun *box* could have the following entry in the lexicon:

(6) box := N_{e7} : box(e7)

Box is defined (":=") as being an N linked to the variable e7. The identity of this variable is arbitrary; all that matters is that each word in the string is given a novel extension variable, to avoid unintended name clashes. The definition above associates a constraint with the noun, indicating that any value assumed by e7 must name a box in the context. Function categories differ only in that they are coupled to *mappings* between variables. Consider the category for a determiner, NP/N: given a noun to the right, this yields an NP. With semantics attached, the definite determiner appears as:

(7) the := NP_{e1}/N_{e1} : unique(e1)

This simply states that the variable associated with the noun is the same as the one associated with the NP, e1. This ensures that the constraint arising from the determiner, *unique(e1),* will restrict the set of candidates that apply to the noun, whatever they turn out to be.

The combination rules of the grammar have responsibility for relating these representations to each other. In relating the above determiner and noun, for instance, the variables e1 and e7 should be identified, because they are really both about the same set of candidate entities. Thus, in addition to the usual cancellation in the syntax, the rules of application and composition must *equate* extension variables. This is achieved by imposing a constraint of equality on the variables concerned, and the application of *the* to *box* appears as the reduction in (8).

(8) $NP_{e1}/N_{e1} + N_{e7} \rightarrow NP_{e1}$: equal(e1,e7)

The constraint of equality expresses the fact that the candidates for e1 must be the same as those for e7; all constraints which apply to e1 must also apply to e7, and vice versa.

Let us now go back to the phrase in (1), and consider the representations bestowed upon its component parts. Prepositions are functions taking an NP argument on their right to form another function, which must modify a noun on the left. They are therefore categorised as (N\N)/NP, where the leftward application of the inner function is indicated by a backward slash '\'. Because of its relational standing, *in* maps between two distinct extension variables:

(9) in := $(N_{e3}\backslash N_{e3})/NP_{e4}$: in(e3,e4)

Thus, e3 will be identified with the variable of the head noun and related to e4, which represents the prepositional object.

Any noun is form-class ambiguous between its primitive category, N, and a *type-raised* category, N/(N\N), a function over noun-modifying functions. Type-raising gives a noun like *rabbit* the opportunity to compose with *in*, categorised as (N\N)/NP. Note that this operation simply alters the noun's syntactic status, and does not complicate the semantics:

(10) rabbit := $N_{e2}/(N_{e2}\backslash N_{e2})$: rabbit(e2)

The means by which the parser decides between the two noun categories is beyond the scope of the present paper; suffice it to say that the choice is made in accordance with "The Principle of Referential Failure" of Altmann (1987). In the following analysis of (1), we will assume the type-raised version of *rabbit*, which expects modifying material, and the primitive version of *hat*, which closes the whole NP.

The constraint arising from a definite article, introduced above as

unique(e1), is different in kind to the constraints arising from *rabbit, hat* and *in.* It is defined as follows:

(11) unique(eN) is true if the candidate set for eN is singleton

Whereas a constraint like *rabbit(eN)* is a predicate over the particular discourse entities which make up the set for eN, *unique(eN)* is a predicate over the set as a whole. As such, *unique(eN)* is a "meta-constraint": it makes a check on the degree to which a given candidate set is constrained. There is another aspect of (11) which distinguishes it from the rest. Whereas *rabbit(eN)* has its effect as soon as the word *rabbit* is read, the constraint in (11) is only enforced when the NP corresponding to the variable eN is syntactically closed.

Finally, a note on parsing. The input string is processed by a non-deterministic variant of a shift-reduce parser, working from left to right in the string and reducing whenever it can. The details of the algorithm are omitted from the following trace, and the interested reader is referred to Pareschi and Steedman (1987), and Haddock *et al.* (1987) for further discussion of parsing issues.

IV *THE RABBIT IN THE HAT*

The table in (12) indicates the successive states of the system as it reads each of the five words in (1) and makes a combination with the previous constituent.

The first move in processing the phrase is to read the category of the initial determiner, NP_{e1}/N_{e1}, and add its condition to the constraint satisfaction process (CSP). This simply asserts that e1 is expected to have a single candidate, when the corresponding NP is closed. (Notice that the constraint is labelled with an asterisk in (12a), because it applies only at the time of syntactic closure.) The second word is now read, as defined in (10). This places a condition of rabbitness on the variable e2, which at present is unconnected to e1. The addition of *rabbit(e2)* prompts the CSP to search for suitable values for e2 in the context, and it finds three entities fitting the description: R1, R2 and R3. The two categories can now be combined, using composition. This cancels N_{e1} and N_{e2}, producing the category in (12b). A constraint of equality arises from the reduction, meaning that e1 and e2 are treated as alternative

(12)	*Category*	*Constraints*	*Extensions*
a.	NP_{e1}/N_{e1}	*unique(e1)	
b.	$NP_{e1}/(N_{e2}\backslash N_{e2})$	rabbit(e2) e1 = e2	e1 : {R1,R2,R3}
c.	NP_{e1}/NP_{e4}	in(e3,e4) e2 = e3	e1 : {R2,R3} e4 : {H1,B1}
d.	NP_{e1}/N_{e5}	*unique(e5) e4 = e5	e1 : {R2,R3} e4 : {H1,B1}
e.	NP_{e1}	hat(e6) e5 = e6	e1 : {R2} e4 : {H1}

names for the same set.

The preposition (defined in (9)) is now processed, introducing two new variables to the CSP. Satisfying the relation therefore produces two new sets of candidates: one for all containing objects (e4) and one for all objects contained (e3). The subsequent composition with the category in (12b) matches e3 with e2. This removes the first rabbit, R1, from the set for e1 (which equals e2) as R1 is not contained in anything. (12c) records the current category and present state of the CSP. Notice that the second hat, H2, is not included in the set of candidates from which the inner NP must later take its reference, because it does not contain R2 or R3.

The second determiner is read, with the category NP_{e5}/N_{e5}. Reducing *the-rabbit-in* with *the* does nothing to further refine the set of containing objects, but the system now expects this particular set to be a singleton when the phrase is complete. As (12d) indicates, the incremental evaluation of *the rabbit in the* has created two distinct sets of candidates, for the two NPs in the phrase. The variables e1, e2 and e3 all denote the set {R2,R3}, holding possible referents for the complex NP, while e4 and e5 specify {H1,B1} for the inner phrase.

The final noun introduces the category N_{e6} and adds the constraint *hat(e6)* to the CSP. The rule of application is used for the first time, closing the phrase, and producing the final category in (12e). Identifying e6 with e5 eliminates the box, B1, and constrains each candidate for e1 (= e2 = e3) to be in the remaining hat, H1. So R3 goes out. We are left with two singleton sets, thus ensuring the success of the two checks for definiteness. The conditions *unique(e5)* and *unique(e1)* are indeed enforced at this stage, because both the simple NP (corresponding to e5) and the complex NP (corresponding to e1) have been closed by the application to *hat*.

V CONCLUSION

A number of aspects of the present research have been put to one side in this paper. Nothing has been said about the treatment of sentences, in contrast to noun phrases, and little on the way in which certain structural ambiguities are resolved by reference to the context. Clearly there are phenomena for which the account sketched above is overly simple. Nonetheless, it is encouraging that a system of incremental evaluation can provide a solution to an instance of anaphora which poses a problem for more conventional approaches.

ACKNOWLEDGEMENTS

I wish to thank Mark Steedman and Henry Thompson for the supervision of this project. The work has also benefitted from the comments of: Gerry Altmann, Einar Jowsey, Ewan Klein, Colin Matheson, Chris Mellish, and Remo Pareschi. I am grateful to Jo Calder, Robert Dale, Jon Oberlander, and the two reviewers for reading and commenting on earlier versions of this paper. The work reported here was supported by SERC research quota award 83317765.

REFERENCES

Ades, A. and Steedman, M. J. (1982) On the Order of Words. *Linguistics and Philosophy*, **4**, 517-518.

Altmann, G. (1987) Modularity and Interaction in Sentence Processing. In Garfield, J. (ed.) *Modularity in Knowledge Representation and Natural Language Processing*. Cambridge, Mass.: Bradford/MIT Press.

Haddock, N. J., Klein, E. and Morrill, G. (eds.) (1987) Categorial Grammar, Unification Grammar, and Parsing. Technical Report No. EUCCS/WP-1, Centre for Cognitive Science, University of Edinburgh, 1987.

Hirst, G. (1983) Semantic Interpretation against Ambiguity. Technical Report No. CS-83-25, Dept. of Computer Science, Brown University, 1983.

Mackworth, A. K. (1977) Consistency in Networks of Relations. *Artificial Intelligence*, **8**, 99-118.

Mellish, C. S. (1985) *Computer Interpretation of Natural Language Descriptions*. Chichester: Ellis Horwood.

Pareschi, R. and Steedman, M. J. (1987) A Lazy Way to Chart-Parse with Extended Categorial Grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1987.

Steedman, M. J. (1985) Dependency and Coordination in the Grammar of Dutch and English. *Language*, **61**, 523-568.

Steedman, M. J. (1987) Combinatory Grammars and Human Language Processing. In Garfield, J. (ed.) *Modularity in Knowledge Representation and Natural Language Processing*. Cambridge, Mass.: Bradford/MIT Press.

Winograd, T. (1972) *Understanding Natural Language*. New York: Academic Press.